2003-09

Manpower requirements database for the Greek Navy

Sergis, Kyriakos N.

Monterey, California. Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

MANPOWER REQUIREMENTS DATABASE FOR THE GREEK NAVY

by

Kyriakos N. Sergis

September 2003

Thesis Advisor:	Daniel Dolk
Second Reader:	Rudy Darken

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Manpower Requirements Database for the Greek Navy			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kyriakos N. Sergis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**
13. ABSTRACT (maximum 200 words) The Greek Navy is trying to create a Web-enabled Data Base system, which will enhance and facilitate the process of assigning duties (jobs) to its officers. This study provides a prototype of implementing the job-to-officers assignment process by creating a manpower Data Base accessed via the Internet. This prototype is based on the 3-tier architecture, having both the Web and Data Base design and implementation. Behind the scenes, is a multi-criteria decision algorithm that takes the officers' credentials and the officers' and commands' preferences into account and then it determines the best distribution of the officers to the available jobs. This thesis and the supporting research will strive to develop the requirements and a working prototype web site for the detailer and reduce both manpower and time required to complete the assignment process conducted by the Greek Navy's Department of Personnel.			
14. SUBJECT TERMS Web-Enabled Database, Relational Database, Manpower Systems, Three-Tier Application, Multi-Criteria Decision Problem, Agorithm, Greek Navy, Officer, Command, Credentials, Qualifications, Officer's Preference, Command's Preference			**15. NUMBER OF PAGES** 341
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MANPOWER REQUIREMENTS DATABASE FOR THE GREEK NAVY

Kyriakos N. Sergis
Lieutenant, Greek Navy
B.S., Hellenic Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
and
MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Author: Kyriakos N. Sergis

Approved by: Daniel Dolk
Thesis Advisor

Rudy Darken
Second Reader

Peter Denning
Chairman, Department of Computer Science

Dan C. Boger
Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Greek Navy is trying to create a Web-enabled Data Base system, which will enhance and facilitate the process of assigning duties (jobs) to its officers.

This study provides a prototype of implementing the job-to-officers assignment process by creating a manpower Data Base accessed via the Internet. This prototype is based on the 3-tier architecture, having both the Web and Data Base design and implementation. Behind the scenes, is a multi-criteria decision algorithm that takes the officers' credentials and the officers' and commands' preferences into account and then it determines the best distribution of the officers to the available jobs.

This thesis and the supporting research will strive to develop the requirements and a working prototype web site for the detailer and reduce both manpower and time required to complete the assignment process conducted by the Greek Navy's Department of Personnel.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Dr. Daniel Dolk for all his instruction and guidance during the progress of this thesis. His knowledge helped me overcome many obstacles I came around during the evolution of it. I would also like to thank Dr. Thomas Wu and Prof. George Zolla who created my interest in the database and 3-tier architecture concepts. Finally, I would like to thank my family because of the support they have provided me during all these years of studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Greek Navy officers are currently assigned to new billets by detailers, much the same way as the U.S. Navy operates. Detailers are subject matter experts who use intuition and experience to match officers with available command billets. However, officer preferences for available billets and command preferences for available officers are not explicitly taken into account. Thus, it is likely that the assignment of officers to billets is suboptimal with respect to "goodness of fit" involving preferences from both the supply and demand sides.

At present, the detailer has no direct on-line access to manpower data for the naval officers and no direct ability to make decisions. The appropriate data, which are the individual officer's preferences, the Command's preferences and the officer's credentials and qualifications, are collected manually, rather than automatically, and then processed by the detailer who is responsible to make the final decisions. The current process requires time and effort for the detailer to make a final decision. Changes and tracking of each job-officer assignments are difficult to accomplish since there is no tool operated specially for that purpose.

The purpose of this thesis is to develop requirements and a corresponding prototype database, decision support system, and web site for the Greek Navy's Manpower Requirements. This work will develop a web-enabled database by which the detailer - the Greek Navy's Department of Personnel (DoP) officer in charge of the job-to-officer assignment process - can view manpower data about the officers of the Greek Navy, view officers' preferences for available jobs and commands' preferences for available officers, and finally exercise a pattern-matching heuristic which provides a straw-man assignment from which he/she can eventually assign the best officers to the most applicable and available jobs-stations allowing him/her to make appropriate, relevant and rational decisions.

B. AREA OF RESEARCH

The area of research for this thesis deals with multi-tiered web enabled databases, the synchronization of distributed databases, and the use of decision support tools. Currently, the Greek Navy is in the planning stages of developing a "Web-Interface" whereby the detailer can view manpower data on the officers of the Greek Navy and assign the best officers to the most applicable, relevant, and available jobs-stations. All the naval officers will have to visit the website and declare their preferences on-line over the Internet, while at the same time the Commands will designate their own preferences for the officers whom they would like to fill their corresponding job vacancies. This effort will replace the current way of managing manpower data. This thesis and the supporting research will develop the requirements and a working prototype web site for the detailer with the objective of improving the assignment process with respect to goodness of fit while simultaneously reducing both manpower and time required to complete the assignment process conducted by the Greek Navy's DoP.

C. RESEARCH QUESTIONS

- What is an appropriate design for the data, model, and user interface components of a decision support system to support the matching of officers with jobs?
- What a multi-criteria, pattern-matching decision model is appropriate for choosing preferred jobs and/or selecting preferred people to fill specific jobs?
- What overall system architecture model is appropriate for integrating database with decision tools in a Web-based environment?

D. SCOPE AND METHODOLOGY

1. Scope

This thesis will provide a single user prototype for the assignment process. It will provide the essentials for designing and creating a database for the jobs-to-officers assignment process and also integrate some kind(s) of multi-criteria decision model(s) with that database. It will not use real data in most, if not all, cases, but rather use fabricated data to show "proof of concept". Moreover the thesis will provide a means for accessing the database via the Internet.

The scope includes:

- Definition and description of the functional requirements of the Manpower Web Site
- Technical description of the ASP scripts written to implement the functional requirements
- Description of a proposed general administration of the web site and local database
- Development of a prototype web site that utilizes a local relational database
- Demonstration of an operational web site on a server. The following items will be the technical products of my thesis work:
    - Set up backend database (SQL Sever 2000) containing a manpower data file
    - Set up a web server (IIS-5) and load appropriate HTML and ASP files
    - Demonstrate User authentication
    - The prototype will demonstrate several different WRITE pages (data update). The thrust of the prototype is to demonstrate that this approach can work in principle, not to program 50-100 ASP web pages in its entirety.

2. Methodology

The methodology used in this thesis research follows:

- Investigate existing manpower assignment models
- Conduct review of IIS-5 web server technology
- Conduct review of Microsoft SQL Server 2000 technology
- Conduct review of Windows XP Professional network administration
- Design Microsoft SQL Server 2000 database
- Build web site containing web pages for the users – Officers, Commands, Detailer
- Build multi-criteria model for job preference and candidate preference
- Implement multi-criteria matching process
- Test produced prototype

3. Assumptions and Limitations

- Assumptions
    - Network Architecture and Server Software. The Greek Navy is more oriented towards the Microsoft software technology. This justifies the use of a Microsoft's product like SQL Server 2000 for this application.
    - Client Software. Virtually all of the desktop computers within the Greek Navy have a Windows-based operating system, usually Windows 2000 Professional (Client).
    - Database. Beyond Microsoft Access available in the Microsoft Office (2000/XP), there is no widely utilized DBMS (Database Management System) within the Greek Navy. Microsoft Access is widely used at the local unit level. Access is an adequate DBMS client/server product for limited functions, but is not appropriate as a backend database for larger scale requirements with greater security needs. The requirements for this database demand a commercial DBMS. As such, I have selected Microsoft SQL Server 2000 mainly for the ease of integration with the Microsoft based networks used throughout the Greek Navy.
- Limitations
    - Data. The manpower web site prototype does not use real data for a variety of reasons. First, the confidentiality of real data is by itself a significant reason for not using it. A second reason is the limited availability of real data. The dispersion of data makes it difficult to be collected and organized. A final reason is that this prototype is implemented several miles away from Greece.
    - Security. Security features of the manpower web site prototype will be addressed in Chapter V. However, the thrust of this thesis and the prototype is a proof of technical concept. Before any actual deployment of the prototype, it would need to be thoroughly analyzed by security experts to ensure that the manpower data being accessed is indeed secure.
    - Scale. The manpower web site prototype developed for this thesis will not address issues related to scale. Any actual deployment of the web site prototype could entail a sizable load (number of connected users) on the web and database server. The manpower web site prototype is being developed on a home computer that has neither the hardware nor software to handle/test heavily web/database traffic. Professional web and database administrators would need to be employed to test the manpower web site prototype.

- Reliability. Reliability is on the other side of the coin of scale. Again, it is beyond the scope of this thesis to analyze and test the reliability of the web and database server with a heavy load. Commercial servers and their software have features that provide for fail-over mechanisms and mirror sites both for the web server and database server.

In order to fulfill the objectives of this thesis, the material presented will be organized in the following manner. Chapter II will cover background material regarding the Greek Navy's Manpower requirements and related works on that subject. Chapter III will address the database design presenting the Entity-Relationship Diagram (ERD) and the final relational concepts. Chapter IV will cover the multi-dimension decision model and the corresponding pattern-matching algorithm. Chapter V will address the system architecture of the prototype providing a description of the programming of the web pages and database queries necessary to support the functional requirements of the prototype. Finally, Chapter VI will present recommendations, conclusions, and further work on the web enabled database.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In order to analyze the requirements and develop a prototype for the Greek Navy web-enabled database, an understanding of the Greek Navy's DoP's process to assign a job to an officer is required. Also, past researches and papers are thoroughly examined in order to suggest ways and methods that will help to solve the problem more efficiently.

A. GREEK NAVY MANPOWER REQUIREMENTS

Currently the DoP is following a rather old fashioned procedure to select an officer for a specific job. This does not mean that the DoP doesn't use current technology in order to help its job perform its job better. The DoP is using proprietary systems like desktop computers, which have W2K Professional as their operating systems. Based on the needs of the Navy the DoP examines the jobs and their requirements. It also examines the qualifications and credentials of the officers. After that it assigns a job to an officer trying to find the best match between them. It tries to find a match beginning from the officers with higher ranks through those with lower ranks.

The whole process, even if it is quite straightforward, it requires significant effort because of the huge amount of data that is dispersed in different places. The DoP personnel have to first collect the data first, and then process it, and this may require significant man months of time. Things may become more complicated when a change must be made. The personnel might have to reexamine the job and the officers and probably collect different data than the ones collected before since the requirements might have changed. Changes and tracking of each job-officer assignments are difficult to be accomplished since there is no tool operated specially for that purpose.

The DoP decided recently that it will take the preferences of the officers for their next job assignment into account. Every officer must complete a form, which contains all the appropriate personnel information such as the officer's identification number, first name, last name, rank, preference and then send it back to the DoP via secure mail. The DoP personnel collect all these forms and use them for the job-to-officer assignment process. The distribution and collection of the forms may last many days or maybe even weeks. Should a mistake be made or could a form get lost, the whole process for that specific form must be reinitiated from scratch.

At present, the detailer has no direct on-line access to manpower data for the naval officers and no direct ability to make decisions. The officers have to send their preferences manually instead of automatically via an on-line intermediary tool. The commands currently do not have the ability to specify preferences for the officers that the commands would like them to occupy the jobs under their command. The appropriate data, which are the individual officer's preferences and the officer's credentials and qualifications, are collected manually, rather than automatically, and then processed by the DoP, with the detailer who is responsible to making the final decisions.

The Greek Navy wants a place where all manpower data related to the Navy officers and commands will be stored. These data include the credentials and qualifications of the officer, the officer's job preferences and the command's preferences for the officers for a particular job under that command. The qualifications of an officer include the languages that he can speak, and past experience that he/she may have for a particular job. Diligence, discretion and secrecy are some of the credentials that an officer may have. Moreover, data such as the rank, the specialty and the minimum sea time required for an officer's rank should be stored.

This thesis will suggest an alternate approach to replace the current way of managing manpower data. It will strive to develop the requirements and a working prototype web site for the detailer and in order to reduce both manpower and time required to complete the assignment process conducted by the DoP.

B. RELATED WORK

In order to determine an efficient approach to this project, sufficient research should be done to documents that tried to find an effective solution to the multi-criteria decision problem of matching officers and jobs.

The meaning of multi-criteria decision problem, in contrast to the one-criterion decision problem, is that there are at least two criteria as variable inputs in the decision problem. In this particular case, the first criterion is the preference of an officer for a particular job. For example the x officer prefers the y job (that belongs to the z command). The second criterion is the command preference for the officers for a particular job under that command. In other words "the y job (that belongs to the z command) specifies a preference for the x officer". The third criterion is the credentials and qualifications of the officers. For example an x_1 officer may be eligible only for jobs y_1 and y_2 but not for job y_3, whereas an x_2 officer may be eligible for jobs y_2 and y_3, but not eligible for job y_1, but he may be more qualified for the job y_3 than x_1 is.

We will examine two approaches to this problem that have appeared in the recent literature. The first adopts agent-based technology as a way of establishing a marketplace for jobs and officers, whereas the second adopts a more traditional operations research optimization approach based upon the assignment algorithm.

The first approach is described in William R. Gates and Mark E. Nissen with title "Two-Sided Matching Agents for Electronic Employment Market Design: Social Welfare Implications [Reference 1]".

The paper describes an exploratory experiment to assess the performance of five alternative employment market designs. These are the following: a. unassisted, b. assisted, c. personnel mall, d. two-sided matching algorithm and e. optimization. In the first two methods, the job-to-seeker matching process is conducted by people. In the remaining methods, this matching is conducted automatically by different market mechanisms. As the name implies, the unassisted condition is used to assess the performance of people performing the matching task with no technological or algorithmic support. In the assisted condition people use a product called Logical Decisions for Windows (Logical Decisions 1993) to assist them with the matching task. The Personnel Mall uses software agents to represent both employers and job seekers, and quasi-prices (i.e., inverse utilities) to represent employer and job seeker preferences. The fourth experimental condition automates the matching task through a two-sided matching algorithm, which is set up to simultaneously consider the preferences of all employers and job seekers. Lastly, the fifth experimental condition automates the matching task through an optimization algorithm, which explicitly seeks to minimize average quasi-price across the entire set of employers, job seekers, or both.

Table 1 summarizes job seeker, employer and total social welfare for each of the experimental conditions. It appears from these results that the optimization approach produces an increase for both the job seeker and employer social welfare, while the combined optimization produces the highest payoff in terms of total social welfare. The latter conclusion illuminates the need of an optimization algorithm that automates the job-to-officer matching process for the current thesis.

Experimental Condition	Aggregate Job Seeker Social Welfare		Aggregate Employer Social Welfare		Total Social Welfare	
Unassisted	$6.08		$6.52		$12.60	
Assisted	$6.13		$6.55		$12.68	
P-Mall – Employer	$6.13		$6.63		$12.76	
P-Mall – Job Seeker	$6.89***	13.4%	$7.01***	7.5%	$13.90***	10.4%
Matching Algorithm	$7.00***	15.2%	$6.84***	4.8%	$13.84***	9.8%
Optimization – Employer	$6.09		$7.48***	14.7%	$13.57***	7.7%
Optimization – Job Seeker	$7.32***	20.5%	$5.96***	-8.5%	$13.29***	5.5%
Optimization – Combined	$7.06***	16.2%	$7.24***	11.0%	$14.30***	13.5%

*** Significant at 99%

Table 1. Social Welfare Per Assignment (Change from Unassisted Control Group).

The second approach, which was conducted by Hemant K. Bhargava and Kevin J. Snoap, is described in "Reengineering Recruit Distribution in the U.S. Marine Corps [Reference 2]. The purpose of this paper is to improve the way that the U.S. Marine Corps' new recruits are distributed to entry-level schools. The system that performs the distribution is called RDdss and uses a computer-based model called RDM. The RDM finds the best distribution by trying to minimize the total number of unfilled seats over all the entire schools. This paper describes some improvements on that system taking into account a variety of additional factors not heretofore considered.

First of all, the desire of the Marine is fulfilled through a contract guarantee called a PEF (program enlisted for), specified during the recruiting process. A PEF establishes which schools a recruit wishes to go to. A second concern in recruit distribution is that the Marine should be checked to see whether he/she is suitable for a specific school. The

Suitability is determined by matching a Marine's qualifications and a school's requirements, described as properties. This is analogous to the Greek Navy preference system we are proposing.

Third, the timing of the distributions is of great significance since schools may have different starting dates whereas Marines are seeking for jobs every week. Any seats left unfilled in classes are a wasted resource.

Finally, since there may be a lack of seats in the only classes for which a Marine is eligible for, or perhaps because a Marine is not qualified for any of the schools consistent with his or her PEF guarantee, there is a possibility that some Marines may be left unassigned in the end.

The new approach develops a penalty function in which week 1 school seats have a disproportionately high shortfall penalty, since seats left empty in these schools will never get filled. Beyond week 1, shortfall penalty is an inverse function of the school's start date. Table 2 shows the penalty for each unfilled seat per days to school start date.

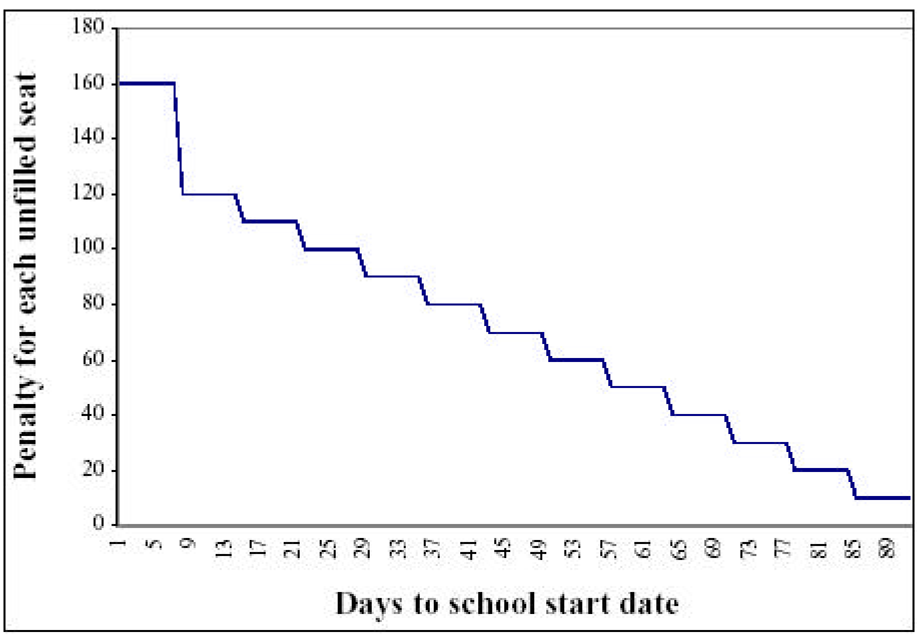


Table 2. Above is the Penalty Function for Not Filling School Seats. The Penalty is Disproportionately High for Week 1 Classes, Since Unassigned Seats Will Remain Unutilized.

RDM was based on a procedure meant to minimize unfilled seats. It is not concerned about the quality of the assignment decisions. There is an obvious tradeoff between the desire to fill more seats and the desire to achieve good fit in the distributions. For that purpose a multi-criteria objective function is used:

$$\text{Maximize Total Utility} = K_{fit} \cdot FitnessScore - K_{fill} \cdot PenaltyScore$$

Coefficients K_{fit} and K_{fill} are control parameters which the model manager can use to create multiple alternative solutions.

In order to compute the FitnessScore, the properties of each school and the qualifications of each Marine are taken into account. Each school has some mandatory properties that affect eligibility. Moreover it may have some desirable properties. These properties are ranked along descending importance in levels 1 through 6.

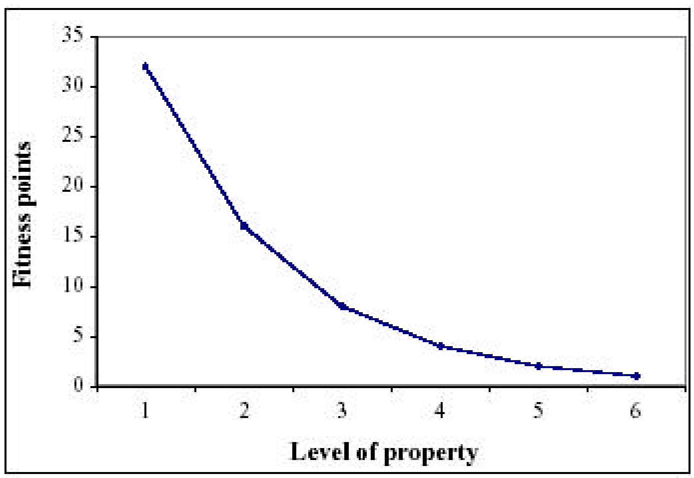


Table 3. Above is the Exponential Function for Assigning Fitness Points Based on the Level of Property Satisfied.

The procedure for computing Marine-to-School fitness can be summarized in two steps as follows.

- For each school, assign a fixed initial score to all Marines who meet the eligibility criteria for that school (ineligible Marines get a score of zero). This score (a typical value is 70) represents the weight given to the mandatory properties in computing suitability. Then, examine desirable properties and assign additional points according to the level of the property to Marines who meet each desirable property (see Figure 3). The result is an initial fitness score for each Marine for the given school.

- For each school, normalize the initial scores so that the average fitness score computed over all Marines eligible for that school is 100. This condition is critical for gaming RDdss to produce good recruitment decisions.

Given the assignment model, it may seem that the solution with the highest utility is the best distribution. However, this is not always true. First, the utility score cannot be used for comparative purposes, partly because it is vulnerable to the choice of scales for measuring penalty and fitness. Second, the relative importance of fit and fill has not been established in the Marine Corps. Third, while fit and fill are important aspects of solution quality they are not the only ones.

For that purpose there are four metrics for evaluating the solution that are defined. The first metric is the total number of unfilled seats in schools starting in the first week. These represent wasted resources. The second metric is the average number of weeks Marines wait before beginning school. The third metric is the total number of Marines not assigned to any school and finally the fourth metric is the fitness premium (averaged over all schools), compared to an average distribution. This is the difference between the average fitness for the proposed distribution and the average fitness (by definition, 100) for an average distribution.

Now it may be seen why it is important to normalize fitness scores. Since all schools have an average fitness score of 100, an average distribution will have a score of 100 for every problem instance. Hence any increase (decrease) in average fitness can be interpreted as a fitness premium that can then be traded off against any loss (gain) in the other 3 metrics. This concept of a fitness premium supports the tradeoff analysis that is necessary to choose a good final solution.

Giving different values to K_{fit} and K_{fill}, different values for the four metrics are produced. Below is a procedure that determines what K_{fit} and K_{fill} values should be used and when the comparison should stop.

- Run the model with $K_{fit} = 0$ and $K_{fill} = 1$. The “fill” and “wait” scores for this run are, by definition, the best achievable fill and wait scores for the given problem instance.

- Run the model with K_{fit} to 1 and K_{fill} to 0. The "fit" score for this run is the best possible fitness for the given instance, and this usually is accompanied by a large loss in fill and wait.
- Set K_{fit} to 1 and K_{fill} to around 10. This run closes part of the fitness gap, but possibly results in some loss in fill and/or wait.
- Conduct additional runs by successively increasing (or decreasing) the fill weight depending on whether the aim is to improve fill (or fit). The final decision is made by comparing the scores on the 4 metrics.

Table 4 gives a representative example of this procedure.

	Run 1	Run 2	Run 3	Run 4	Run 5
K (Fit,Fill) =	(0,1)	(1,0)	(1,1)	(1,6)	(1,4)
Unfilled Seats (wk 1)	***23***	38	25	23	**23**
Average Fitness Premium	28	***34***	33	29	**33**
Unassigned Marines	3	3	3	3	**3**
Average Wait (weeks)	***1.3***	1.7	1.4	1.3	**1.32**

Table 4. Example: Finding a Good Distribution. The first two columns represent extreme solutions on Fit and Fill. Run 3 achieves excellent Fit, but at some loss in Fill and Wait. Run 4 makes only a marginal improvement in Fitness. Run 5 achieves an excellent fit while keeping the best scores on the Fill and Wait metrics.

The methods and concepts of both papers were the guide and directive, on which this thesis' multi-criteria decision model is built. The first paper makes an in depth research over labor market economics and information systems. It conducts five experimental conditions and it considers social welfare as a metric to measure the effectiveness of each one of the experimental conditions. This paper illuminates the need to create and develop an optimized two-sided matching tool and algorithm as it is described through the optimization experimental condition. What is different from the paper is the metrics that this thesis uses. This paper does not provide any clues over the design and implementation of such an algorithm, or any clues about the nature of the two-sided matching tool.

Many of the principles that are used in this thesis are based upon the results of the second paper. The Marines-to-schools distribution concepts are quite similar to those of the jobs-to-officers. One difference is that on the Marines-to-schools distribution model there is a tolerance for having seats unfilled in the end. However, this is not the same case for us. The algorithm should take care of this issue and provide the maximum number of filled jobs. This means that there is no need for having a penalty function concerning unfilled jobs. On the other hand it is necessary to provide priorities to the jobs in order to fill available jobs with the most suitable officers. In other words, the job of the Chief of the Navy must have higher priority than the Fleet Commander and the latter job must have higher priority than the Commanding Officer of a Frigate, and so on.

Moreover, the algorithm for this thesis must take the officer's suitability for a job into account. Every matching of a job with an officer is assigned a value that refers to the degree of fitness between the job and the officer. This value is a number that describes the officer's preferences for that job, the command's preferences for the officer to occupy that job and finally the officer's credentials. Each one of these criteria may have different importance. This importance is measured by a coefficient, just like the K_{fit} and K_{fill} coefficients that are used in the paper. These coefficients are actually weight factors that multiplied by the corresponding criteria values give a weighted estimation of the criteria importance. Again, these coefficients are used as control parameters through which the model manager can create multiple alternative solutions.

A major difference between this thesis and [2] is that the latter considers a utility function as a way to find different distributions and also evaluate them post facto. This thesis uses a greedy-choice algorithm in order to find a distribution. The need for a utility function is based upon being able to evaluate the impact on the solution from any change(s) the detailer may decide to make.

Since the algorithm tries to fill up the maximum number of jobs by always following the same pattern, a change to each coefficient value is not going to affect or change the jobs the algorithm selects. The jobs are always the same. Only the fitness values change, depending on the coefficient values. By changing the coefficient values, the distribution of the officers to the jobs is changed. This means that there is no need to use the various metrics described in [2]. Any change on the coefficients is made on an experimental basis. The utility function estimates how much “worse” off the change the detailer makes is in contrast with the solution the algorithm produces.

Before we implement any pattern matching algorithms, we must first establish an appropriate database design to hold the necessary data for the detailer to evaluate any assignment. The next chapter discusses this database design.

III. DATABASE DESIGN

The data that are stored in the database reflect the needs and the purpose of this project. The database should store an officer's personal information such as his/her name, phone and address, a job's information and information about the platform or base that this job belongs to. It must also contain the credentials and qualifications of an officer and the qualifications that a job requires from an officer in order to be eligible to get that job.

A. REQUIREMENTS

In order to design an appropriate ERD and create a suitable database for this thesis, it is necessary to define the requirements. These requirements are derived from specific queries that the users of the database/website should perform in order to do their job. These queries are the following.

- Who are the officers that participate in the job-to-officer distribution? What is their personal information (e.g., address, phone number or email) in order to contact them?

  This query presents the need of a special place to store personal information such as the first, last and middle name of the officer. Also, the address including the street and city the officer lives in should be provided. The different phone numbers and email addresses the officer has should be stored too. Below is an example from this project's database.

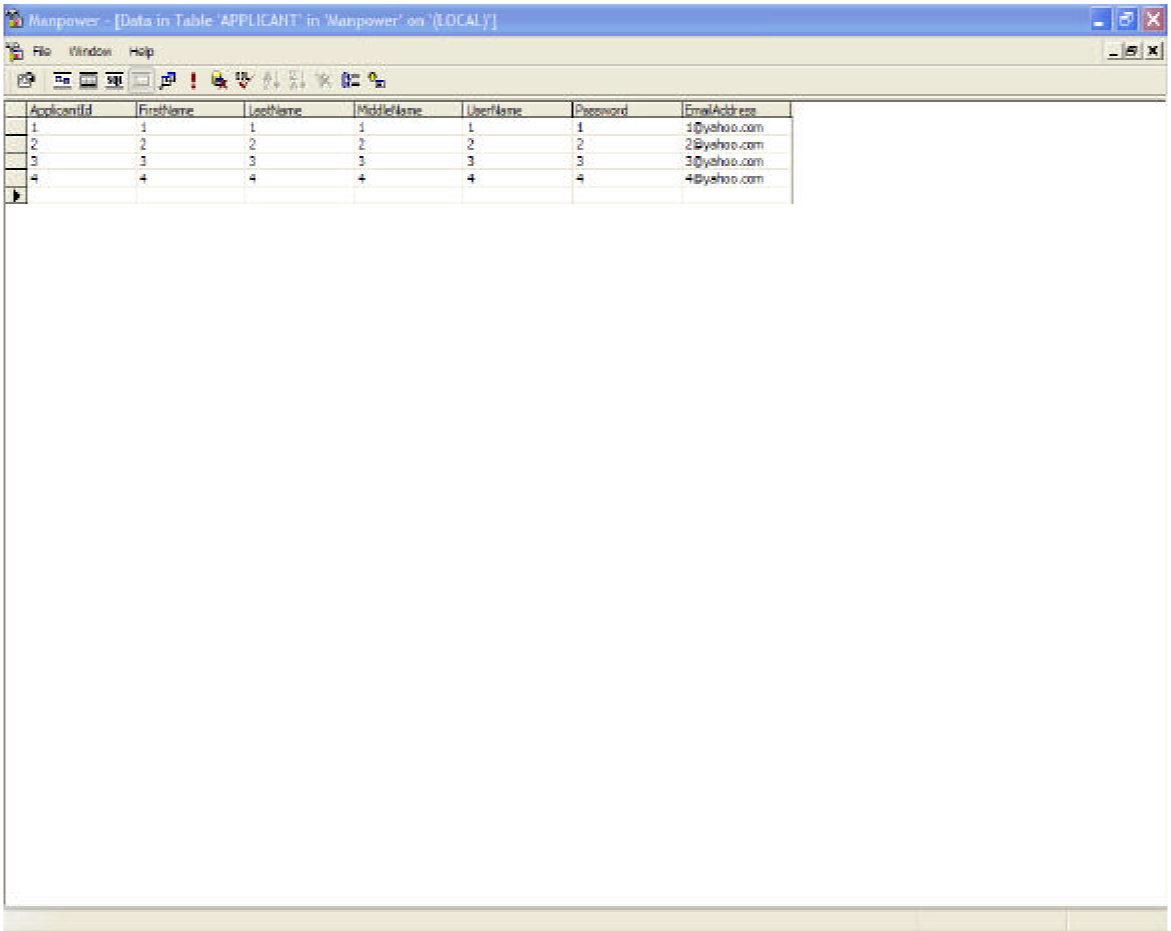


Figure 1. Officer's Personal Information-Manpower Database.

- Who is a valid user for the database/website? What is the username and password of each of the database/website users?

  The officers' and commands' usernames and passwords should be stored too, in order to accept valid users only for logon to the services that the website/database provides. The figure below shows the various usernames and passwords for the Manpower database.

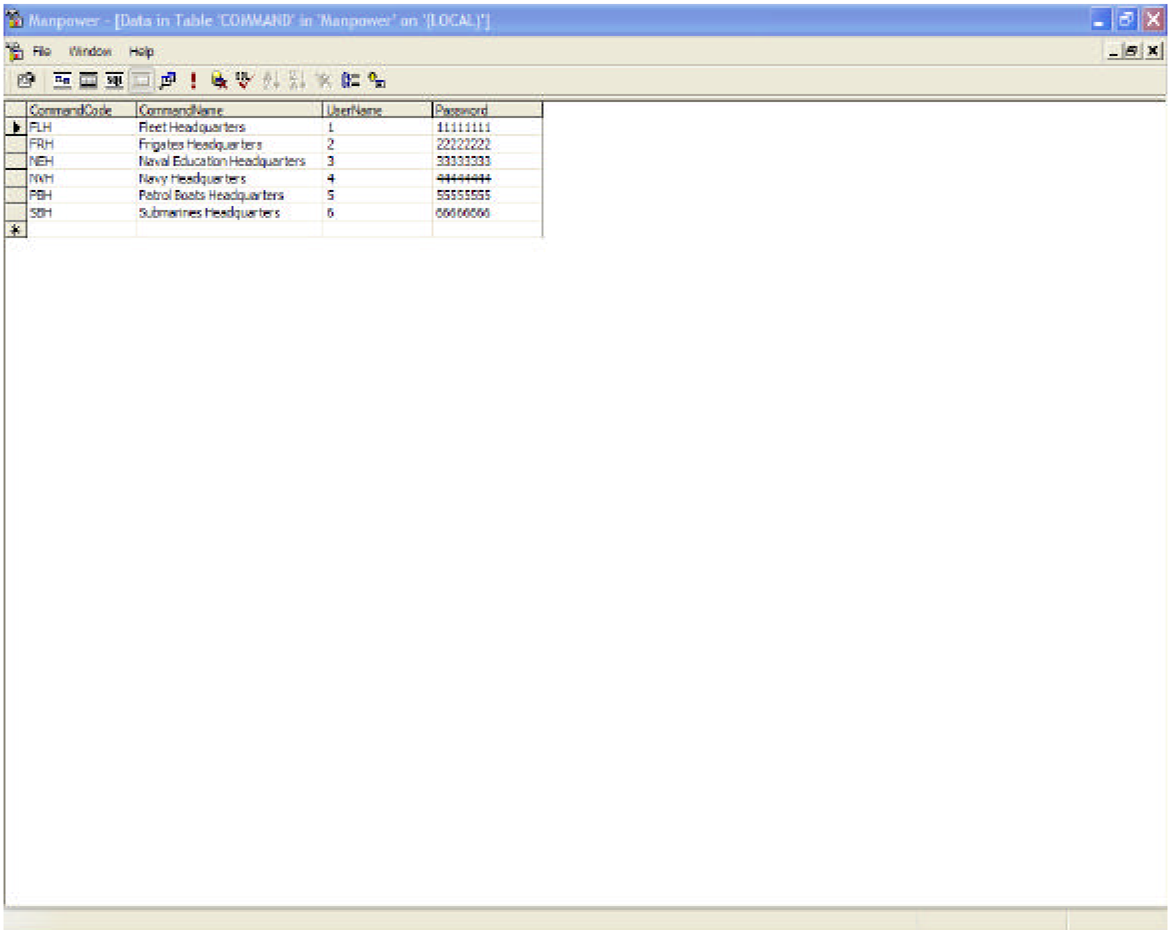


Figure 2. Command's Username and Password-Manpower Database.

- What are the Navy's jobs to which officers may be assigned? Since a job could exist on many platforms or bases (for example the Navigation job exists in all the ships of the Greek Fleet), which are the Navy's platforms/bases the Navy?

  An entity should be created in order to store all the available jobs the Navy has. Moreover, all the available platforms/bases should be stored in a separate entity as well. Also, since a job can exist in many platforms/bases (like the example just mentioned), or a job can exist in some platforms/bases and not in others (for example the Base Commander does not exist in any of the Fleet's ships but exists in all the Navy's bases), there should be a place to store the jobs per platform/base. The figures below show some examples of all these just mentioned.

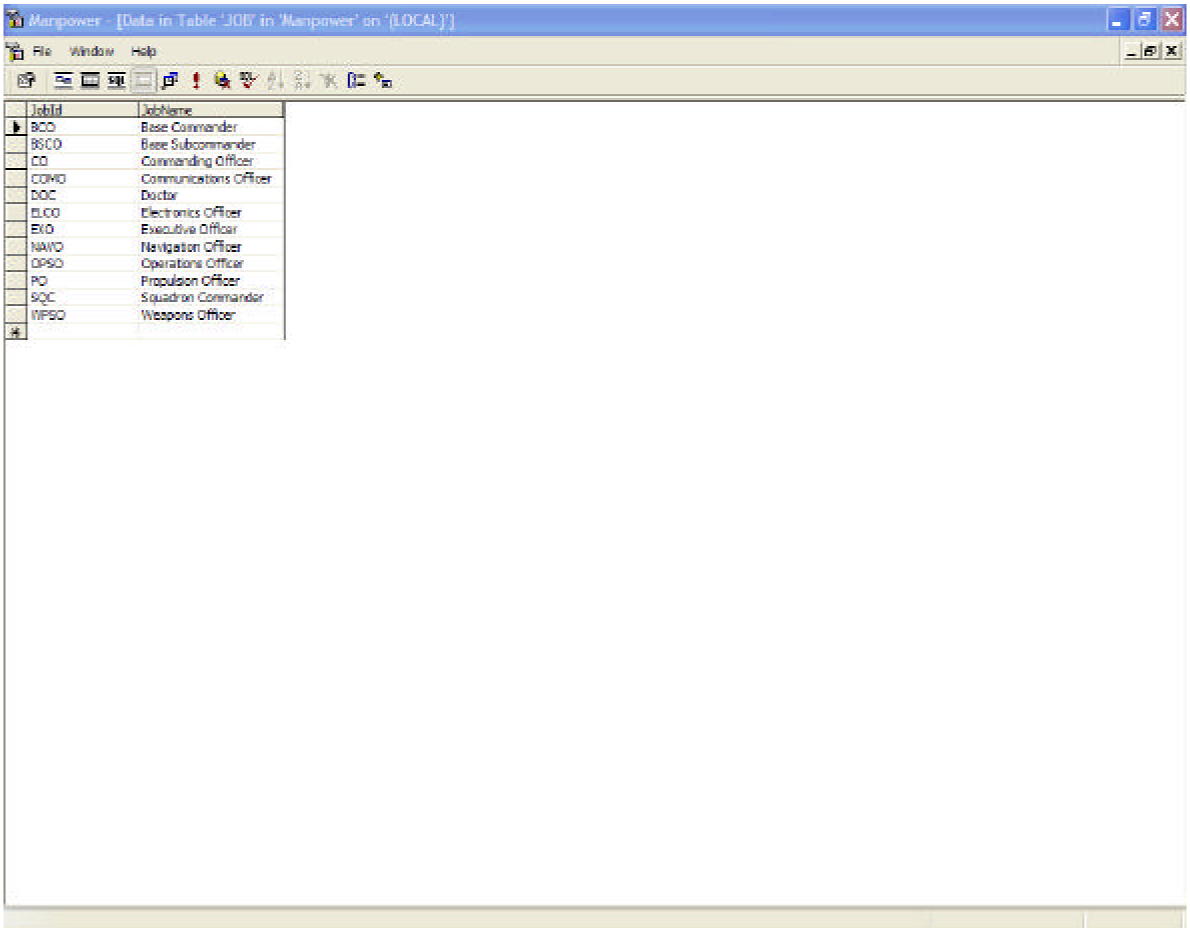


Figure 3. Available Jobs-Manpower Database.

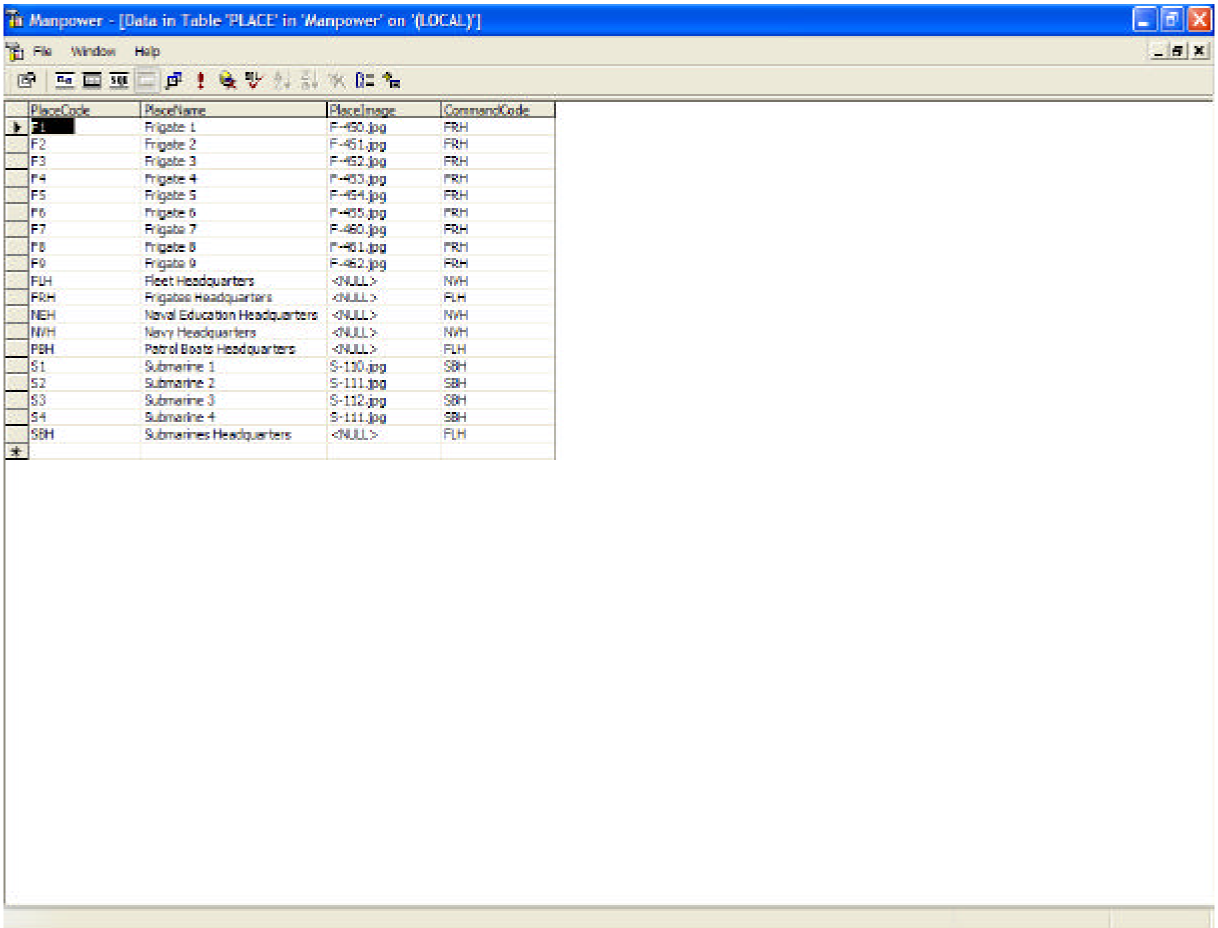


Figure 4. Available Platforms/Bases-Manpower Database.

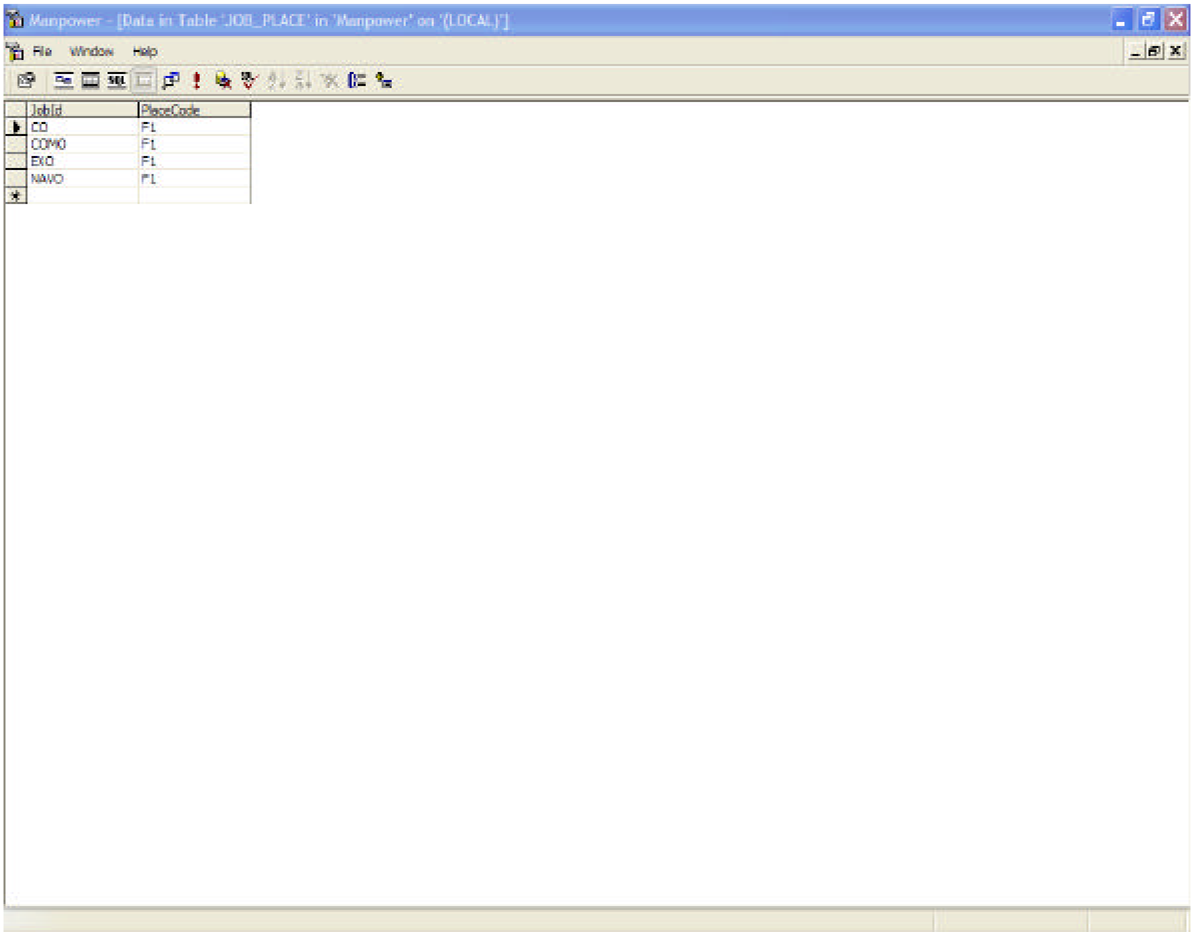


Figure 5. Available Job – Platform/Base Pairs-Manpower Database.

- Which officers are eligible for which jobs? What ranks must an officer have in order to be eligible for a job?

  An Ensign should never be able to be assigned to the Chief of the Navy job. This means that first there should be a place to store all the available ranks, and there should be a place to store the ranks that are required for a specific job (for example a Commanding officer could be either a Commander or a Captain). Third, the rank of each officer should be stored too. The figures below show corresponding examples.

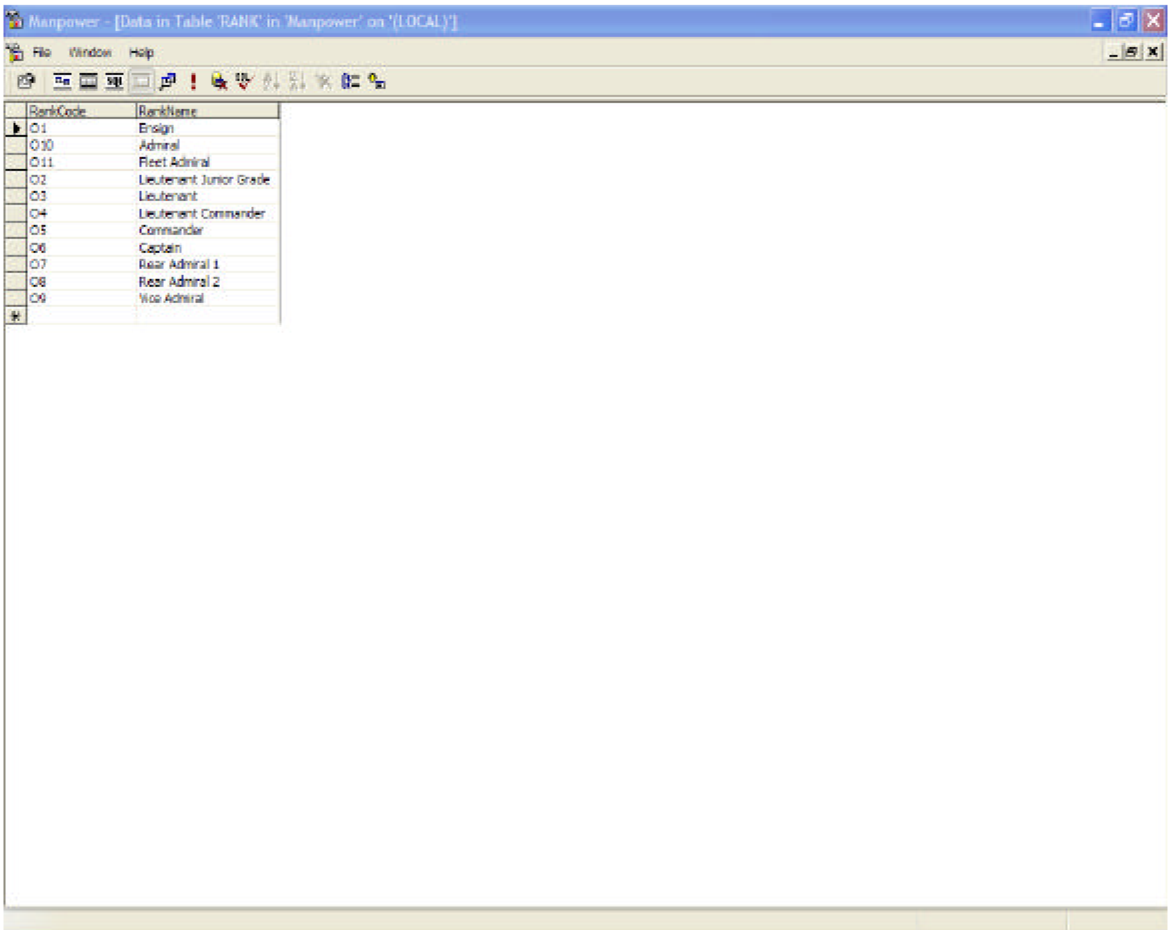


Figure 6. Available Ranks-Manpower Database.

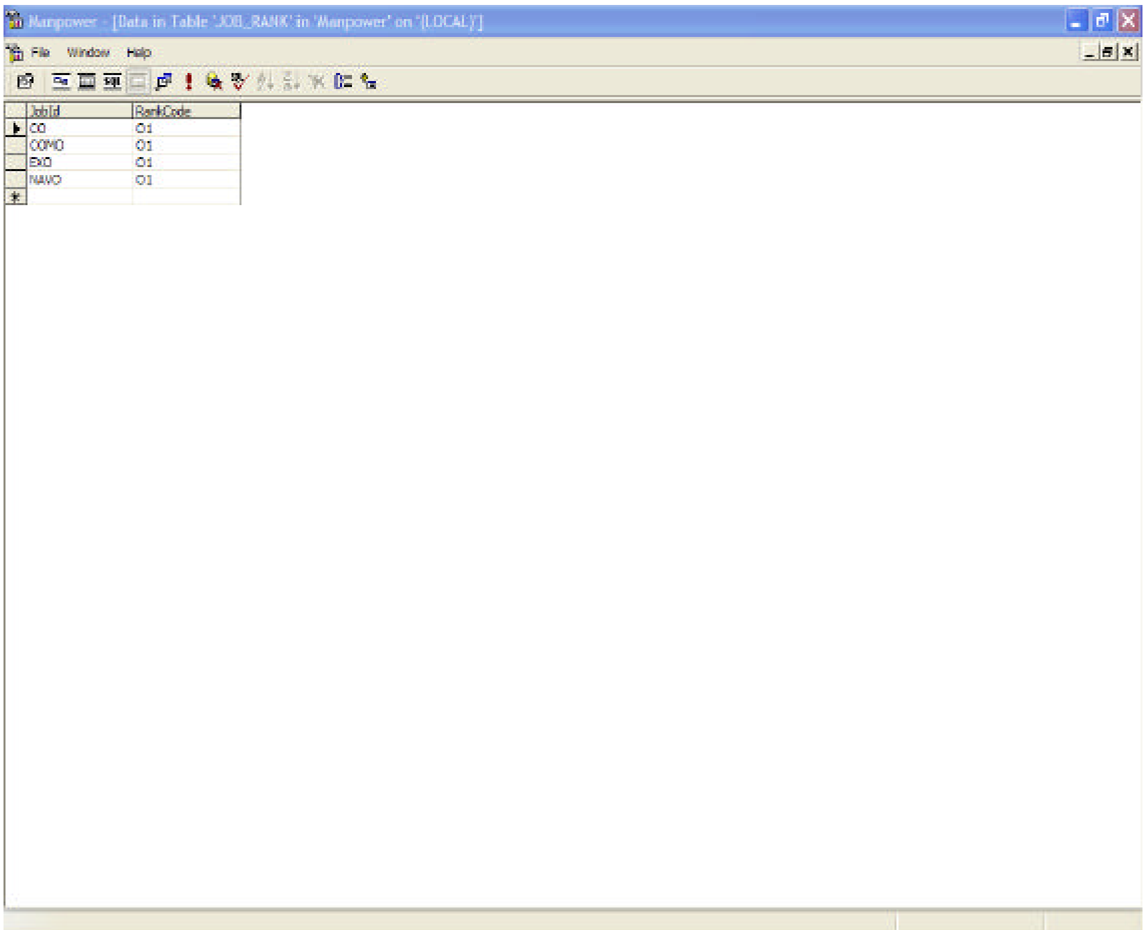


Figure 7. Ranks Required for Different Jobs-Manpower Database.

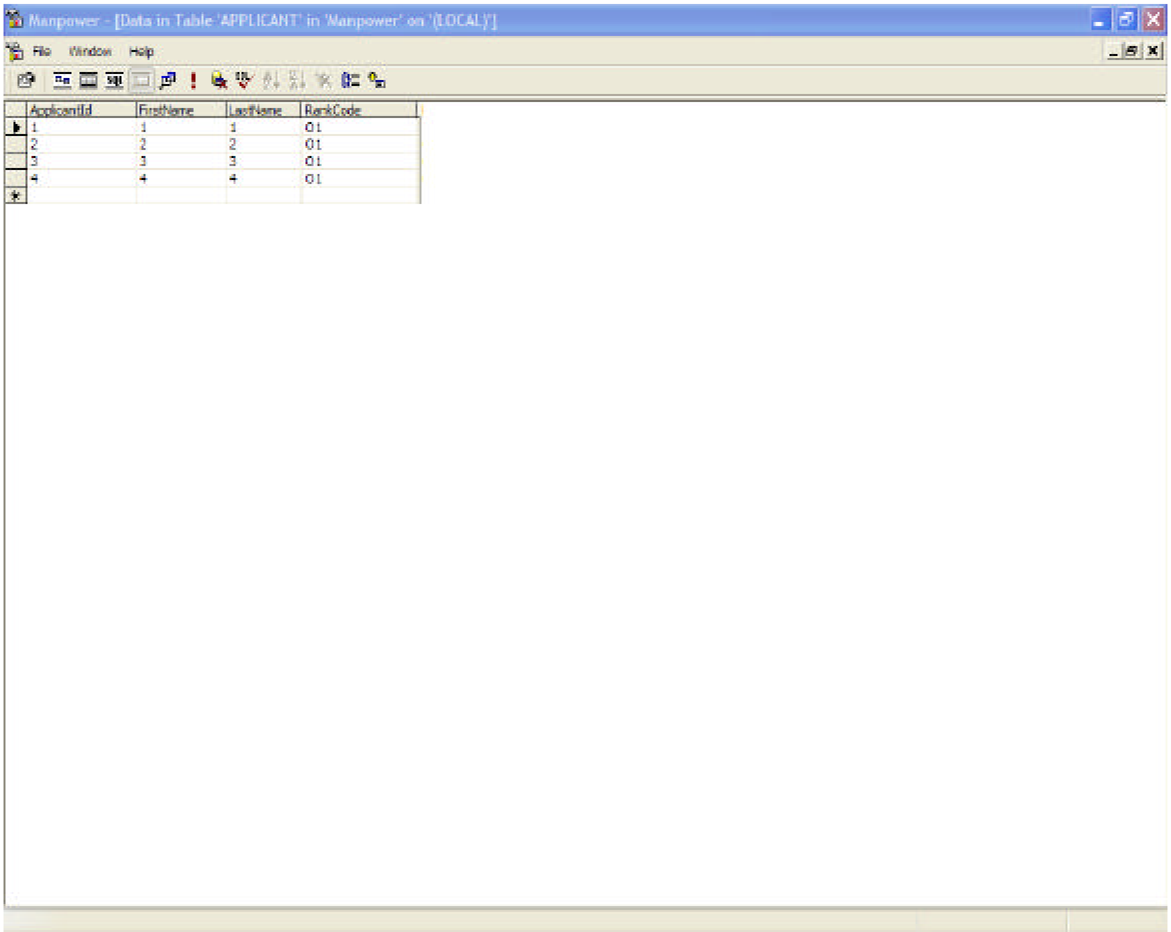


Figure 8. Officers' Ranks-Manpower Database.

- Which specialty should an officer have in order to be eligible for a specific job?

  An officer should be able to get assigned to a specific job, according to his/her specialty. For example an officer should have the Navigation specialty in order to be assigned to the Navigation job for a ship, so there needs to be an entity that describes all the specialties. Also, there should be an entity that describes the specialties each job requires. Moreover, an officer's specialty must be stored too. The figures below show examples.

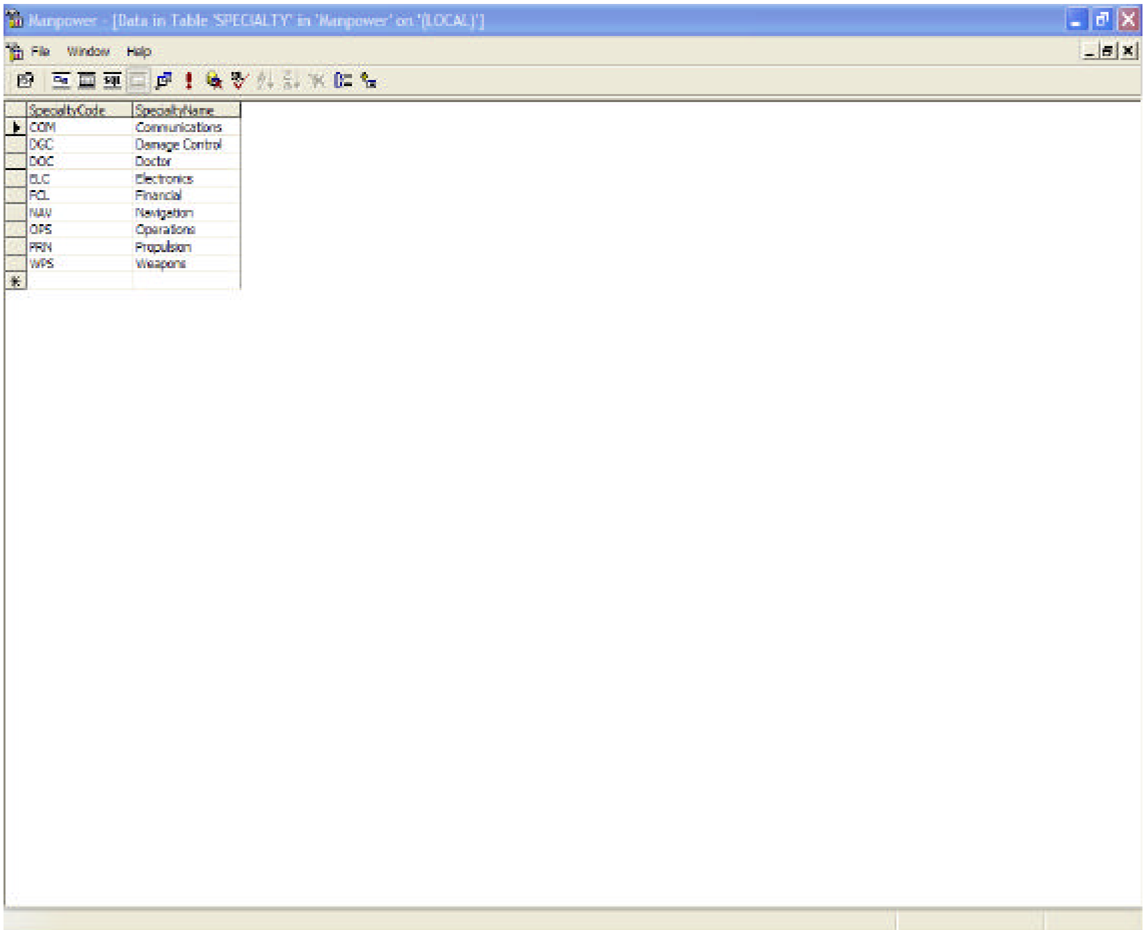


Figure 9. Specialties-Manpower Database.

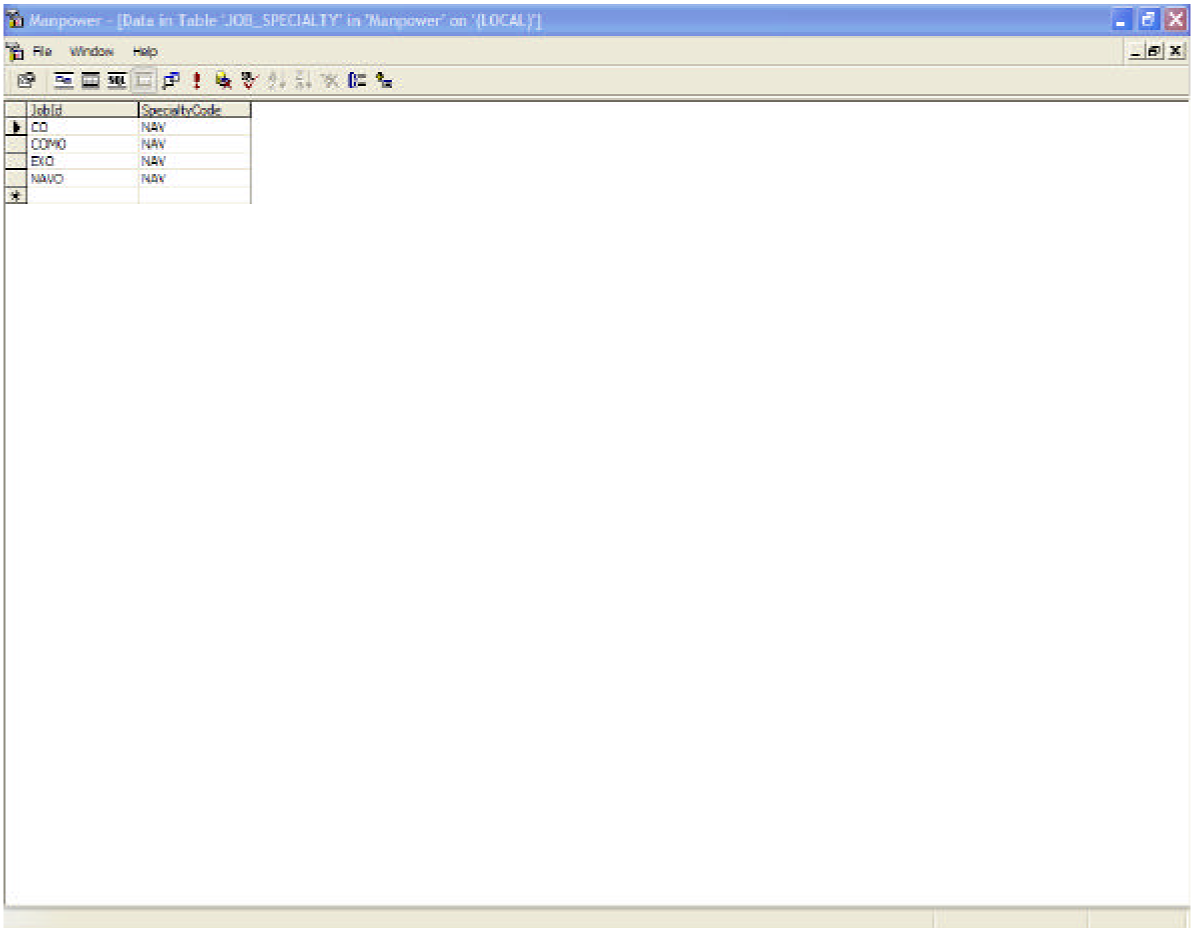


Figure 10. Specialties Required for Each Job-Manpower Database.

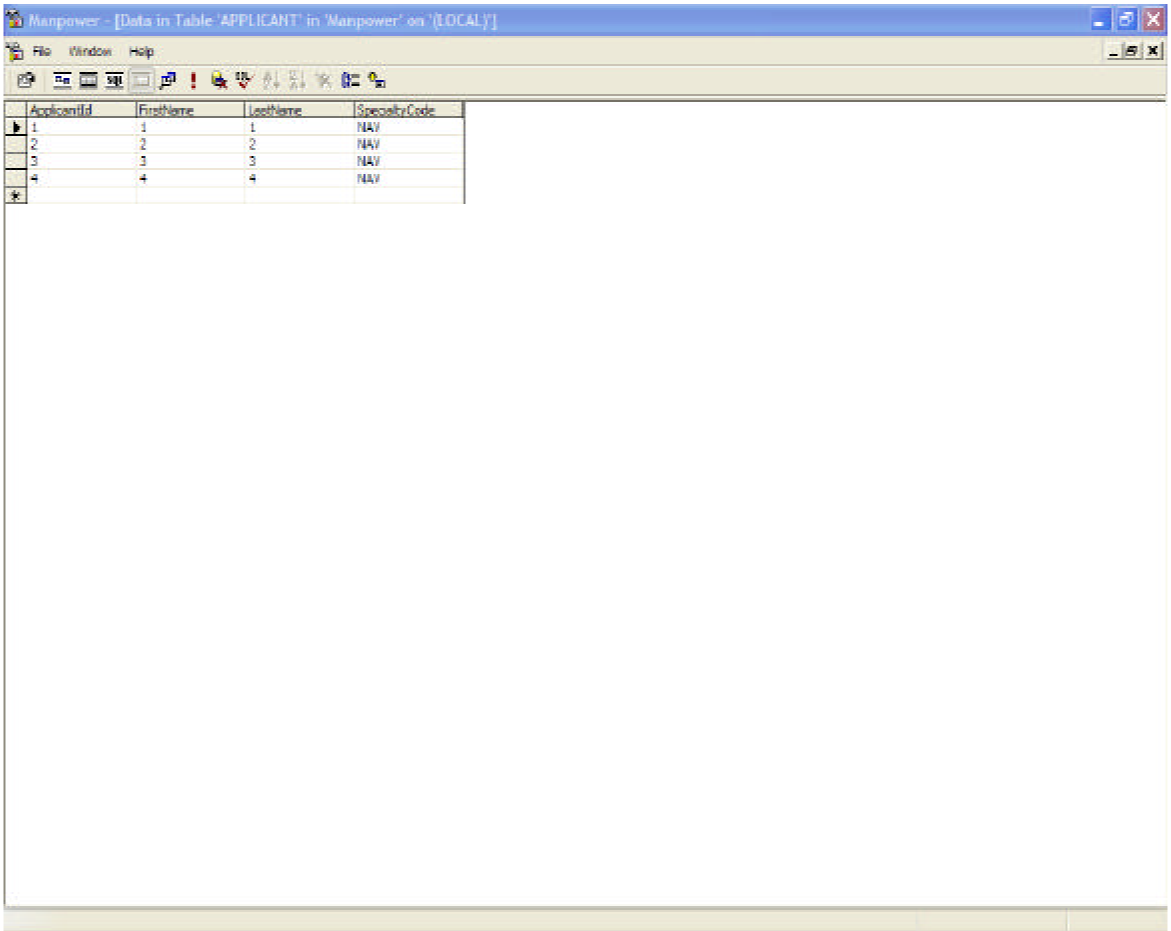


Figure 11. Officers' Specialties-Manpower Database.

- What is the education type an officer must have in order to be eligible for a specific job? Which education type does an officer have?

  An officer's education type is one of the criteria for assigning an officer to a specific job. An entity must be created that contains the whole set of education types, and another entity must describe the education that an officer requires for a specific job. The officer's education type must be stored too. The figures below show pertinent examples.

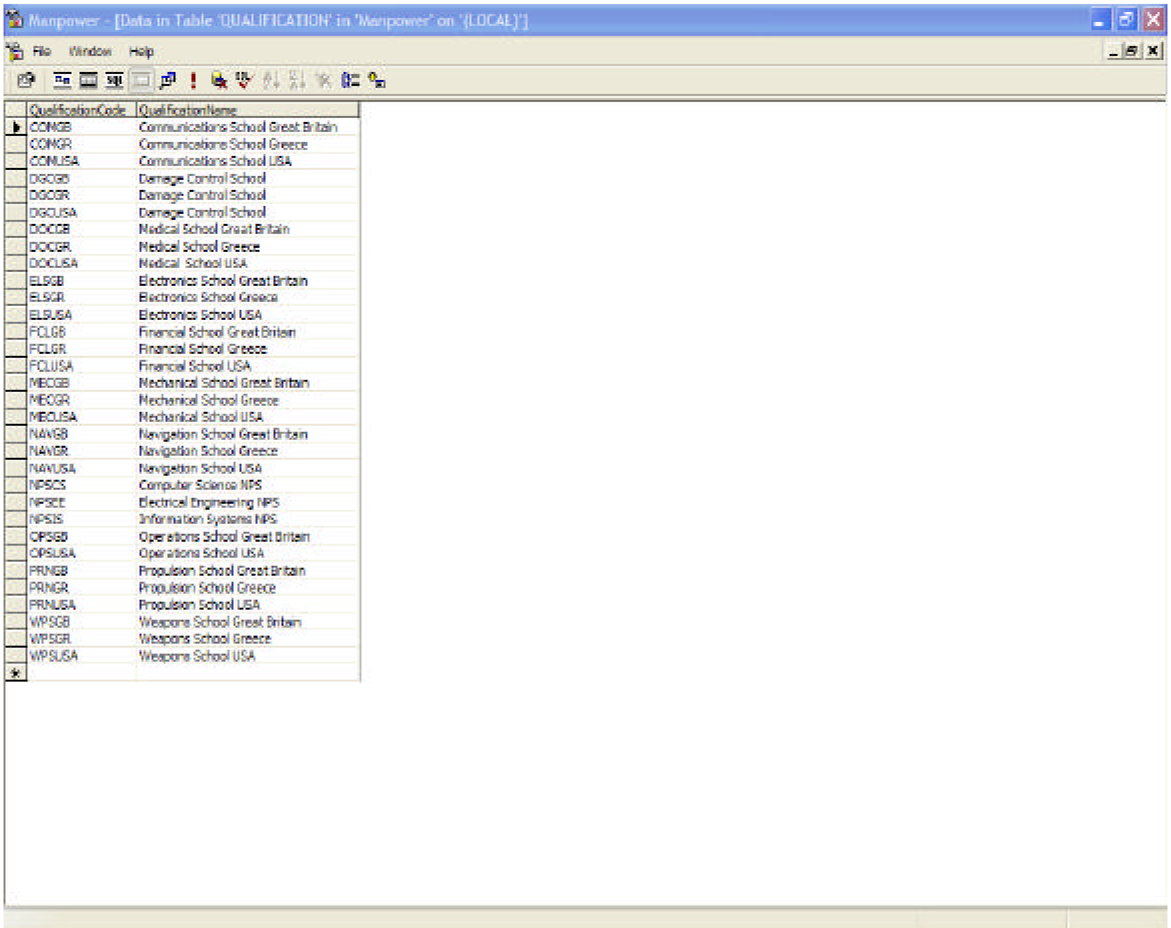


Figure 12. Education types (Qualifications)-Manpower Database.

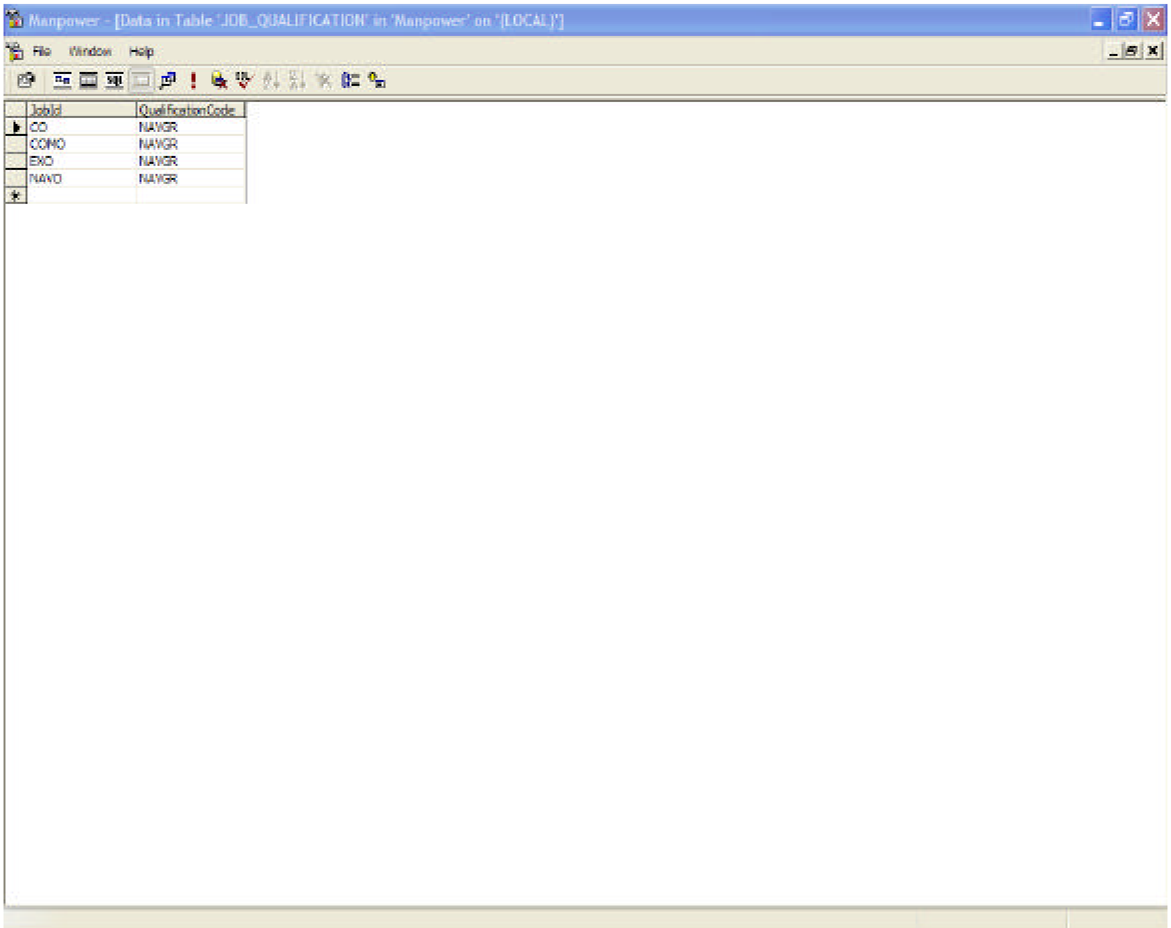


Figure 13. Education Types (Qualifications) Required per Job-Manpower Database.

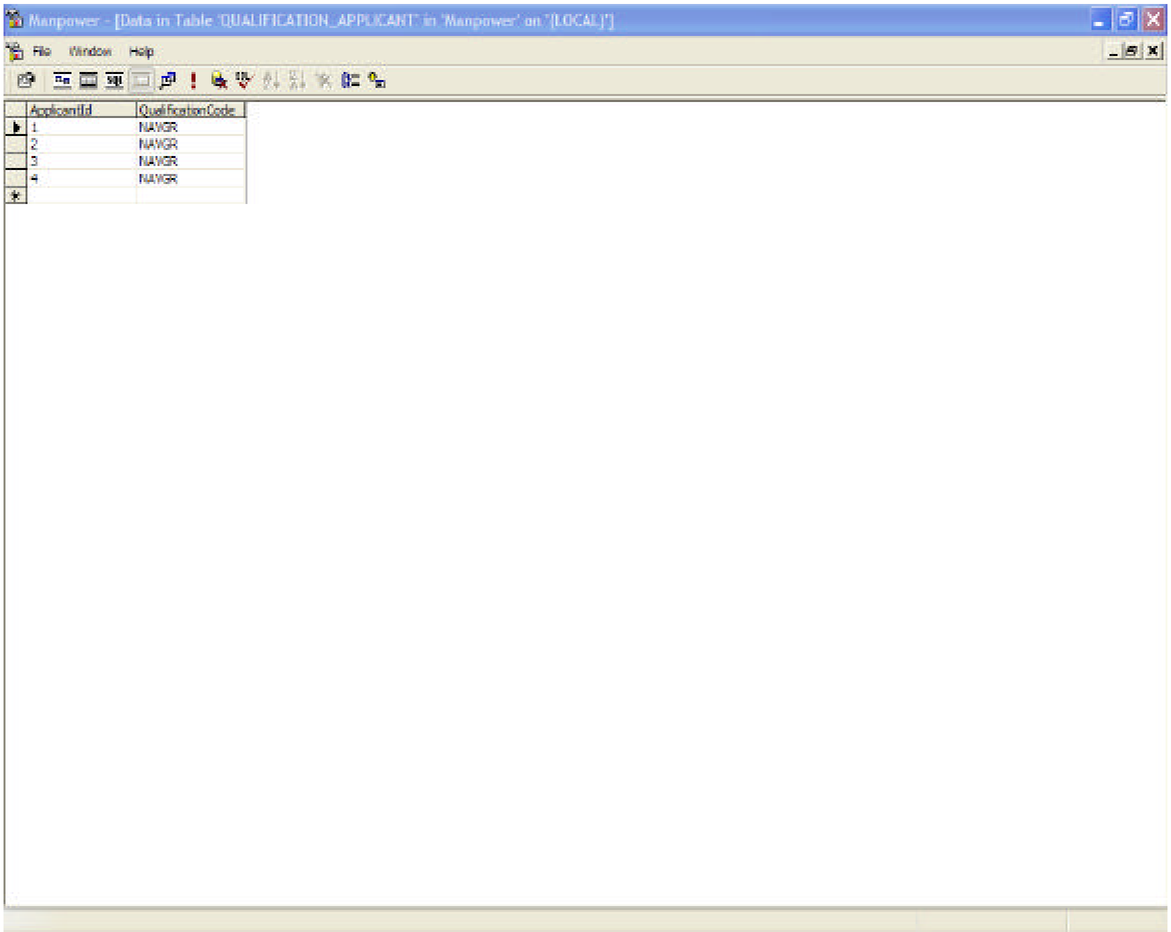


Figure 14. Officers' Education (Qualifications)-Manpower Database.

- What are the attributes an officer must have for a specific job? What is the accepted level of each attribute for an officer to be assigned to a specific job? What are the attributes and levels for each one of the officers?

  Diligence, bravery, and discipline are some of the attributes an officer should have for a job. An entity must be created to store all the available attributes. Also, the minimum level of these attributes for each of the jobs must be stored too. Another entity is required to describe the level of attributes each officer has. The figures below show these examples.

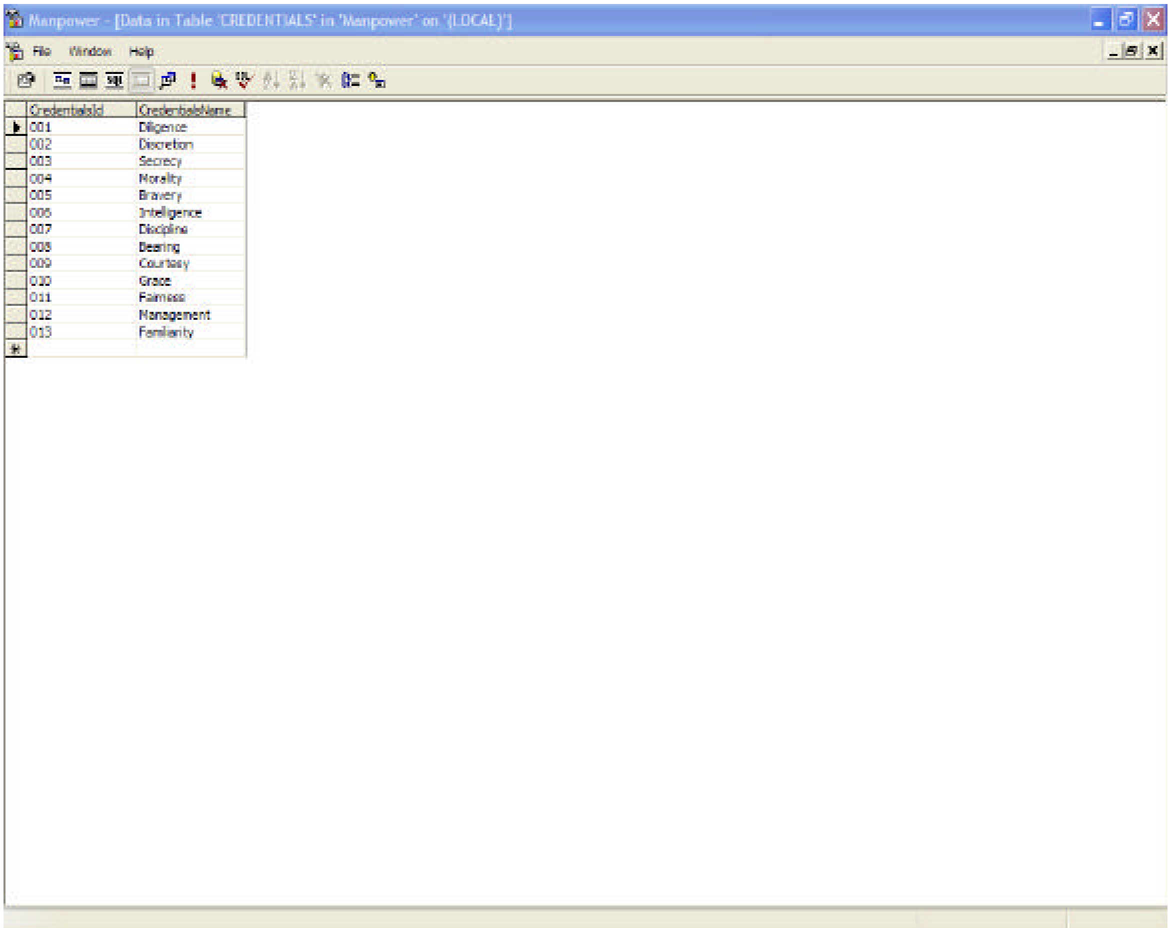
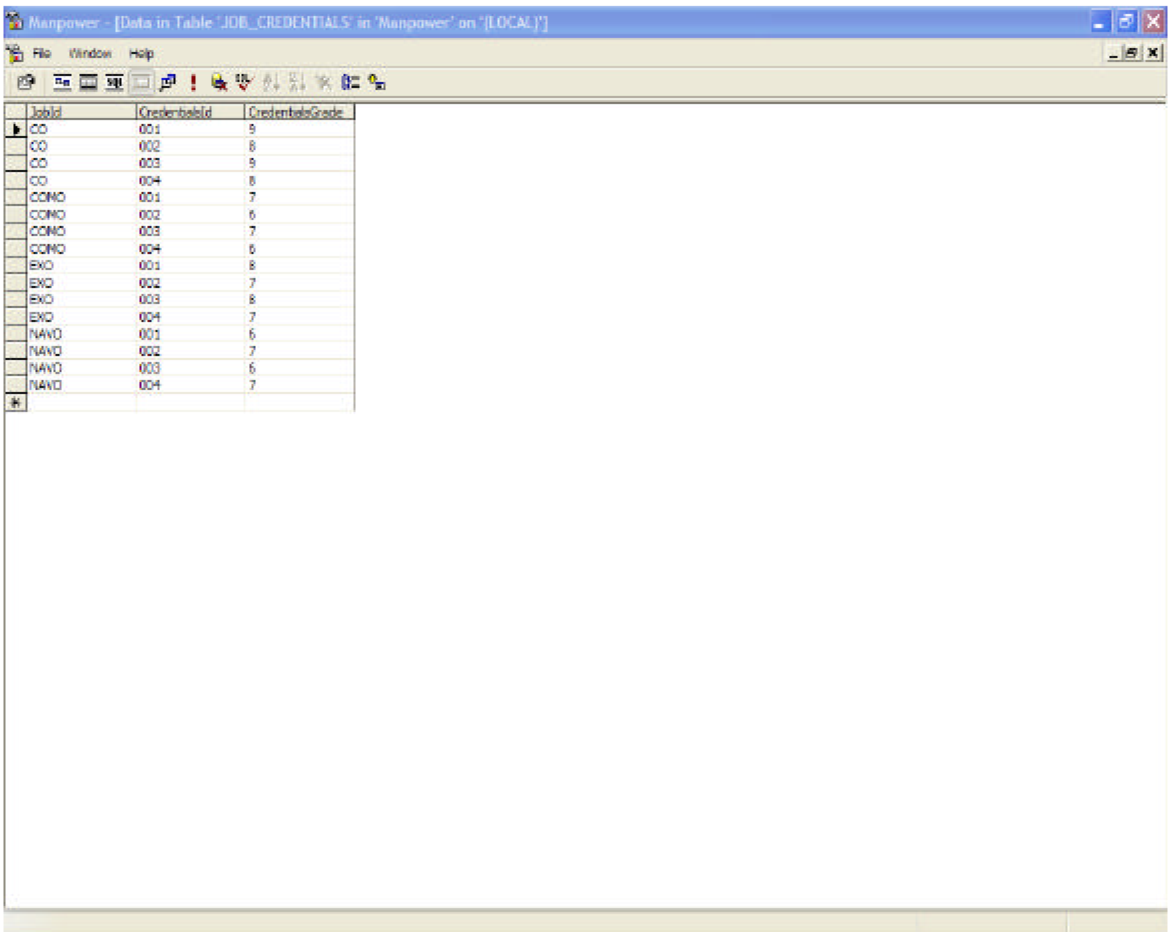
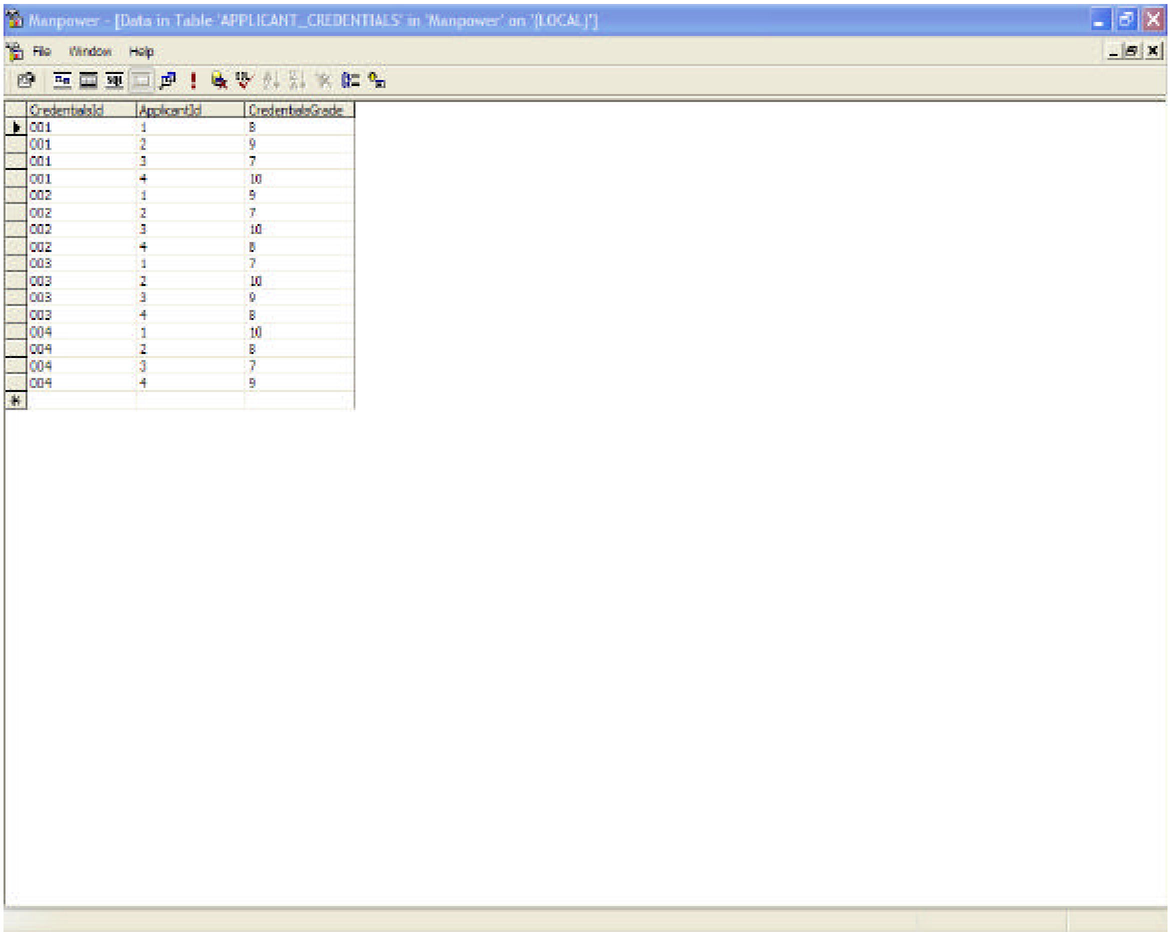


Figure 15. Attributes (Credentials)-Manpower Database.



JobId	CredentialsId	CredentialsGrade
CO	001	9
CO	002	8
CO	003	9
CO	004	8
COMO	001	7
COMO	002	6
COMO	003	7
COMO	004	6
EXO	001	8
EXO	002	7
EXO	003	8
EXO	004	7
NAVO	001	6
NAVO	002	7
NAVO	003	6
NAVO	004	7

Figure 16. Attributes (Credentials) Required Per Job and Corresponding Minimum Levels-Manpower Database.



CredentialsId	ApplicantId	CredentialsGrade
001	1	8
001	2	9
001	3	7
001	4	10
002	1	9
002	2	7
002	3	10
002	4	8
003	1	7
003	2	10
003	3	9
003	4	8
004	1	10
004	2	8
004	3	7
004	4	9

Figure 17. Officers’ Attributes (Credentials) and Corresponding Grades-Manpower Database.

- What are the languages an officer should speak in order to be applicable for a job? What are the minimum levels of these languages for a specific job? What languages and at what level does the officer speak?

  English and German could be language requirements for the job of the Greek Naval Attaché in Germany. All these languages should be stored in a special entity created for that purpose. Also the languages that are required for a specific job should be stored too, with their corresponding minimum levels. Finally, the languages an officer can speak must be stored too, as the figures below show.

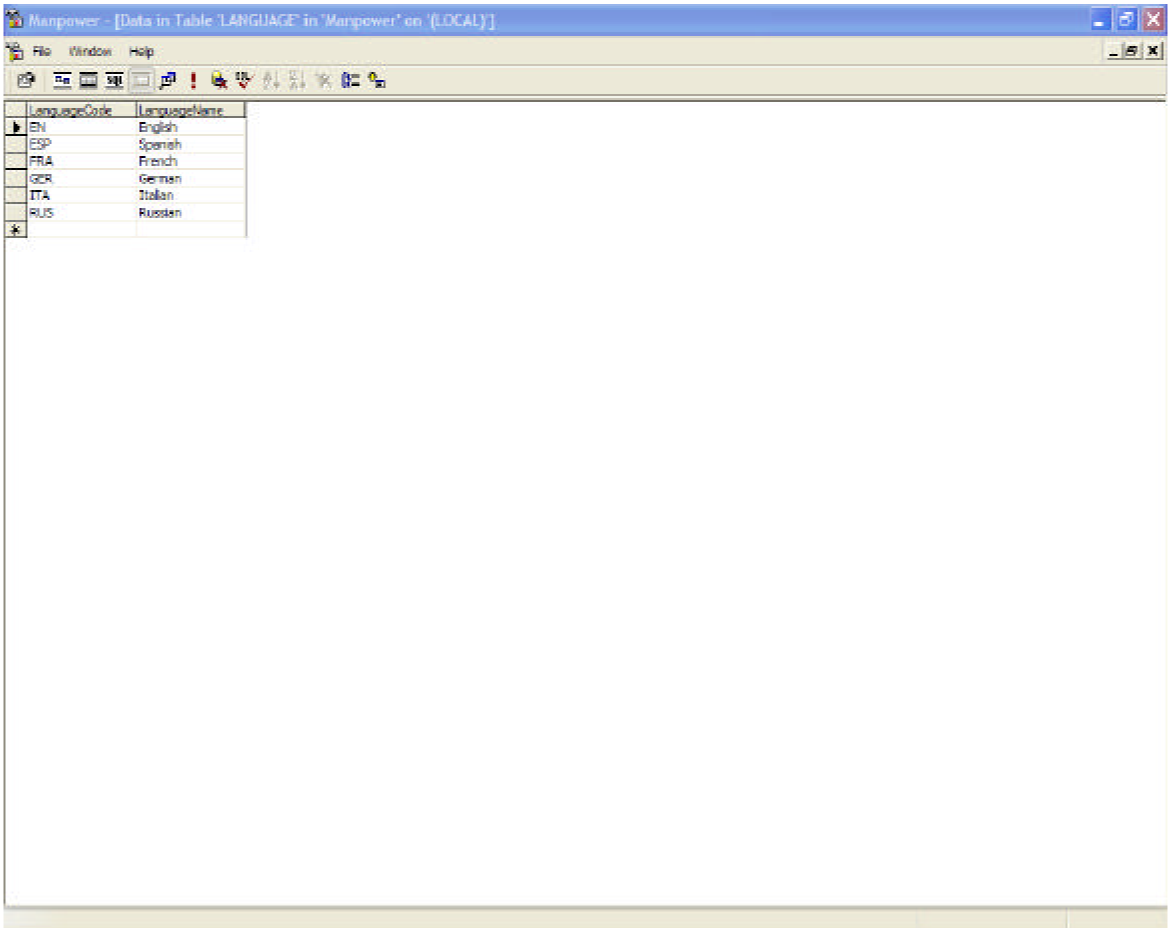
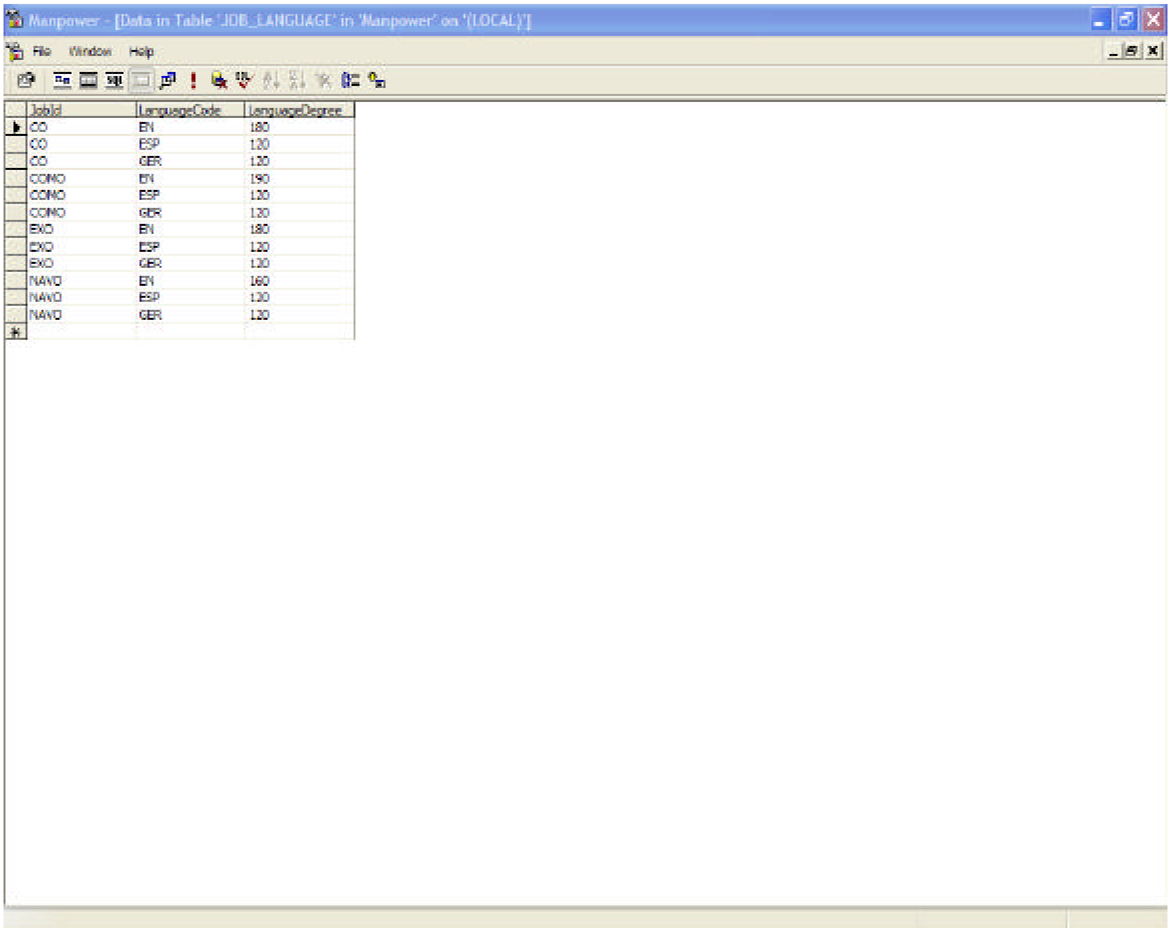
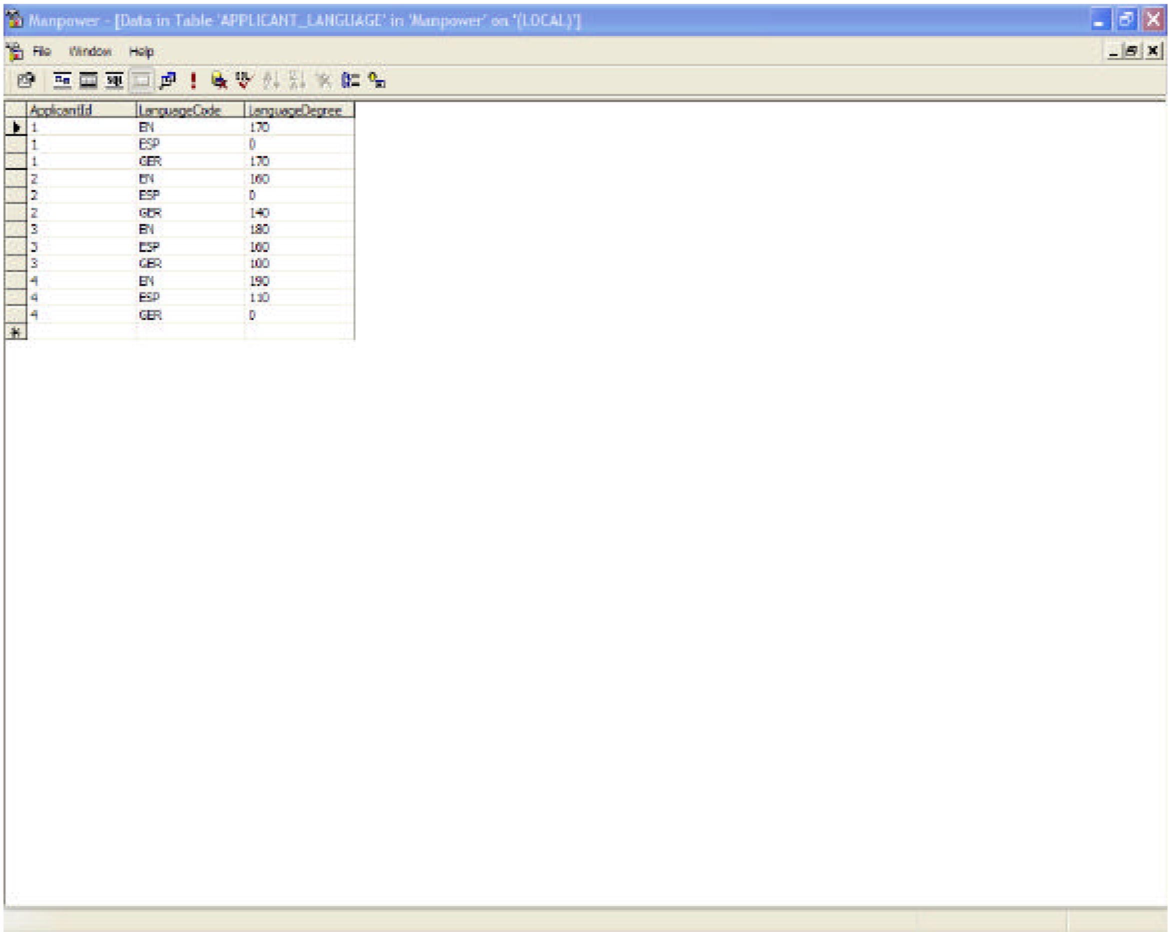


Figure 18. Languages-Manpower Database.



JobId	LanguageCode	LanguageDegree
CO	EN	180
CO	ESP	120
CO	GER	120
COMO	EN	190
COMO	ESP	120
COMO	GER	120
EXO	EN	180
EXO	ESP	120
EXO	GER	120
NAVO	EN	160
NAVO	ESP	120
NAVO	GER	120

Figure 19. Languages Required Per Job and Corresponding Minimum Levels-Manpower Database.

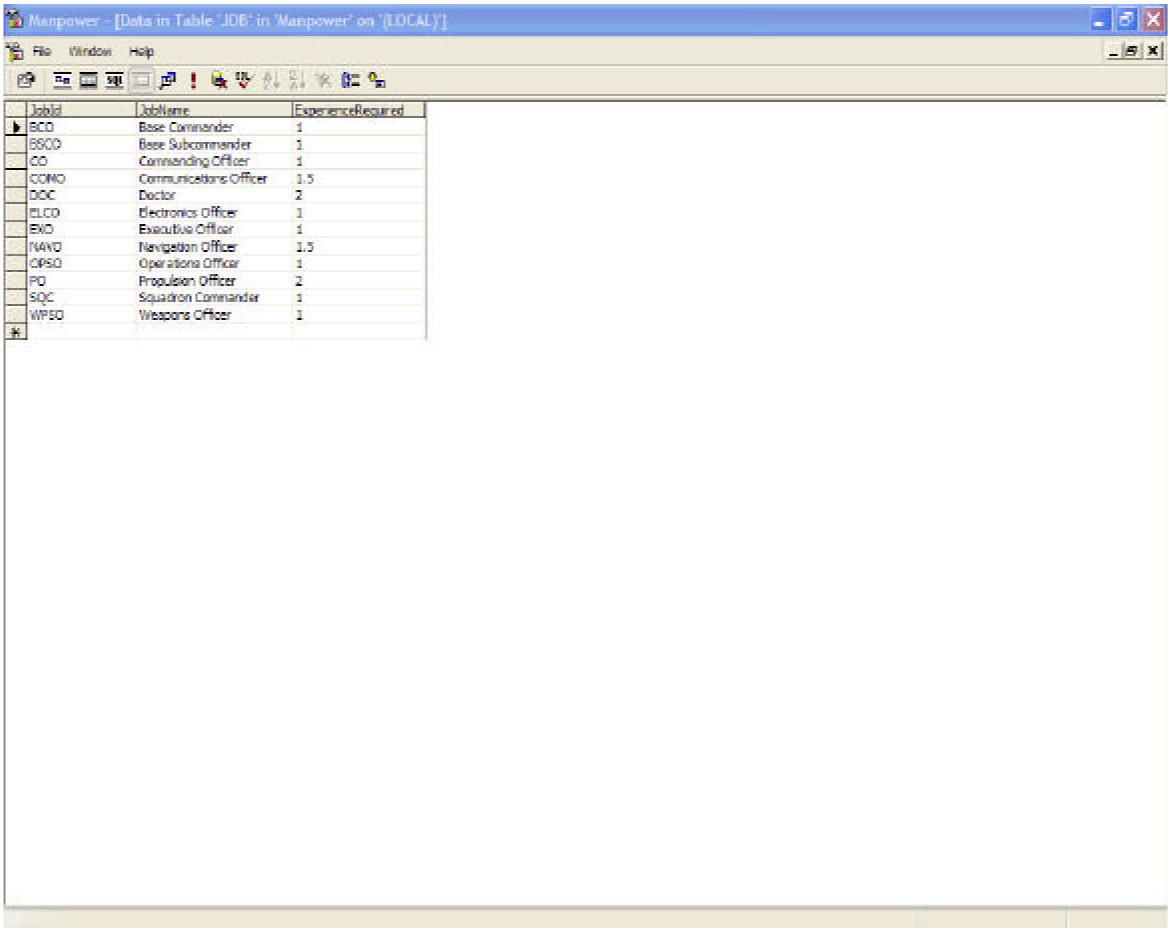


ApplicantId	LanguageCode	LanguageDegree
1	EN	170
1	ESP	0
1	GER	170
2	EN	160
2	ESP	0
2	GER	140
3	EN	180
3	ESP	160
3	GER	100
4	EN	190
4	ESP	110
4	GER	0

Figure 20. Languages Officers Can Speak and Their Corresponding Grades-Manpower Database.

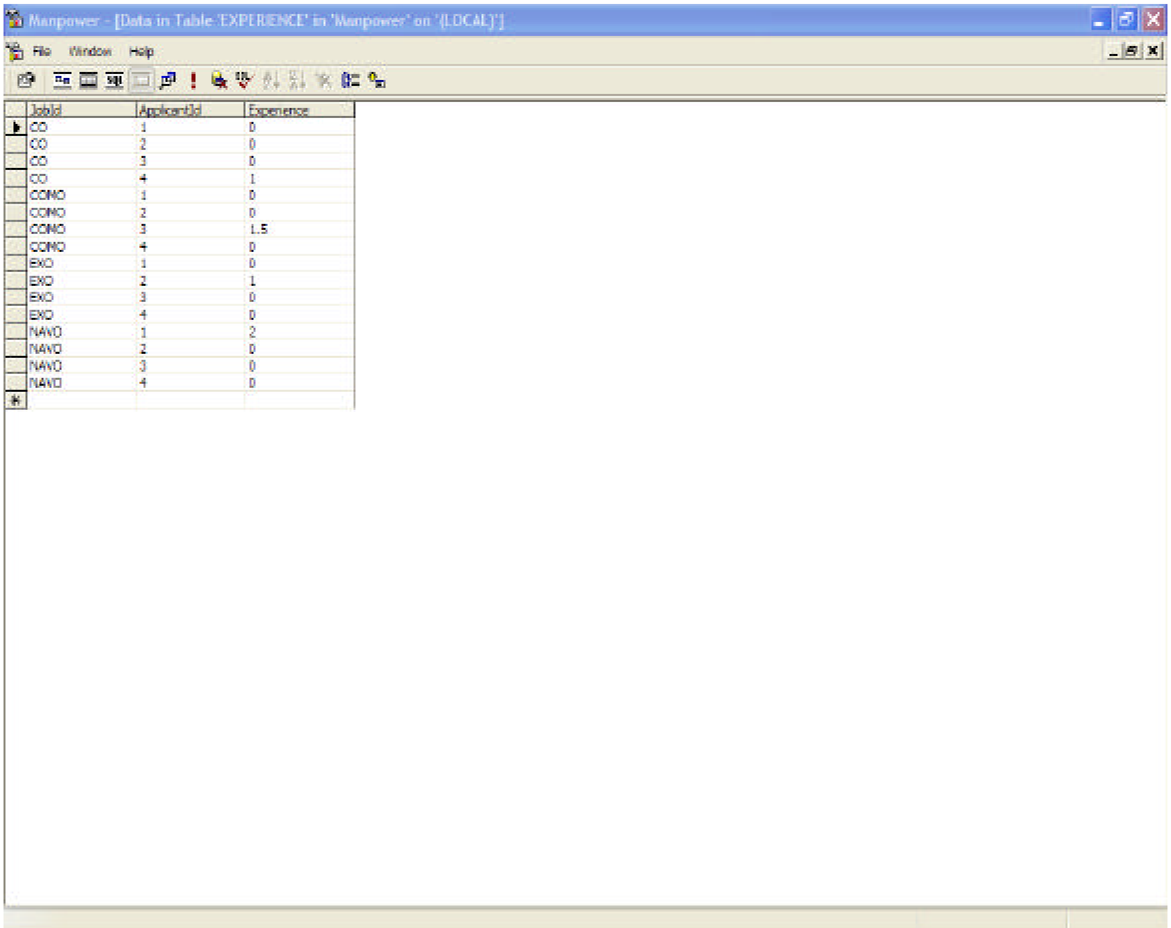
- Can an inexperienced officer be eligible for a job? What are the acceptable levels of experience an officer should have for a job?

  The database should store the years of experience a job requires an officer to have. It should also store the officer's experience. For example, in order to be a Navigation officer, somebody must have at least 1 year of ship experience (see Figures 19 and 20).



JobId	JobName	ExperienceRequired
BCO	Base Commander	1
BSCO	Base Subcommander	1
CO	Commanding Officer	1
COMO	Communications Officer	1.5
DOC	Doctor	2
ELCO	Electronics Officer	1
EXO	Executive Officer	1
NAVO	Navigation Officer	1.5
OPSO	Operations Officer	1
PO	Propulsion Officer	2
SQC	Squadron Commander	1
WPSO	Weapons Officer	1

Figure 21. Experience Per Job Required in Years-Manpower Database.

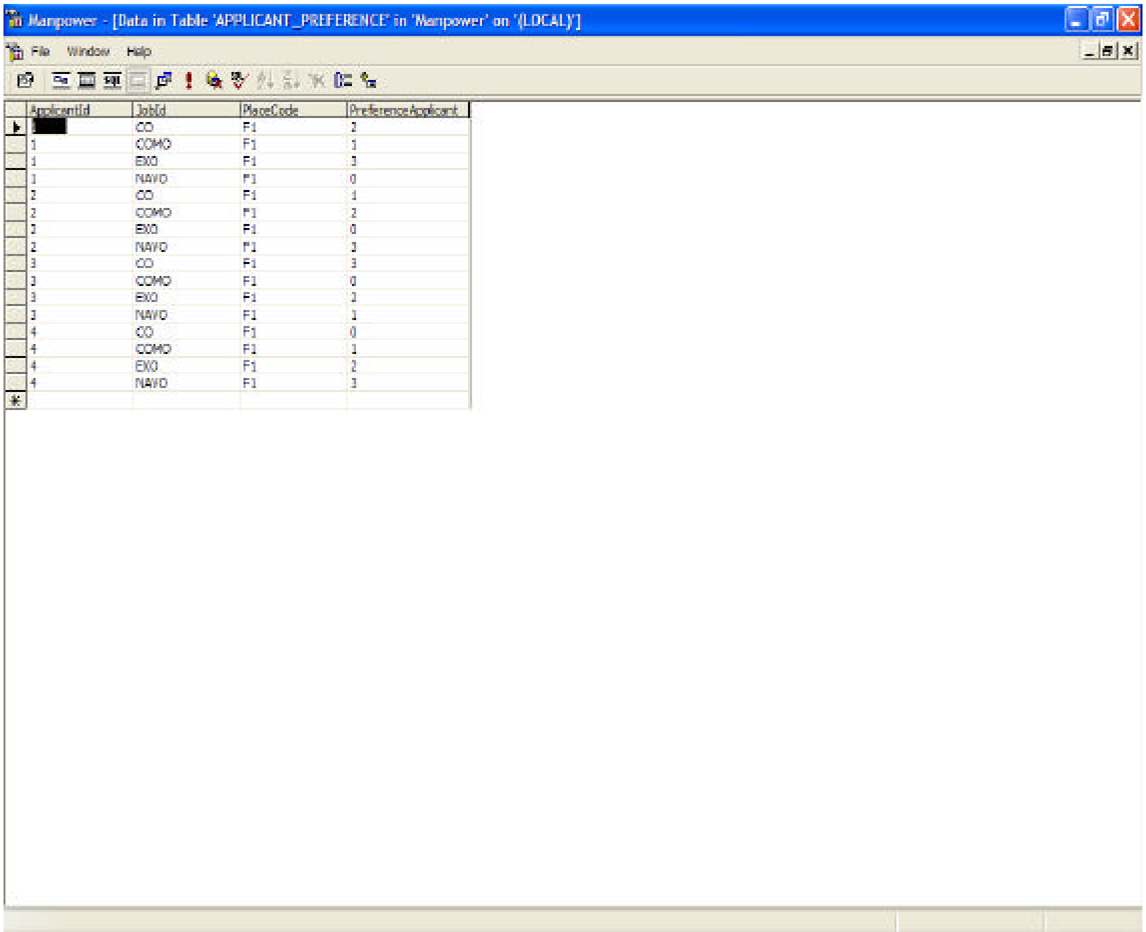


JobId	ApplicantId	Experience
CO	1	0
CO	2	0
CO	3	0
CO	4	1
COMO	1	0
COMO	2	0
COMO	3	1.5
COMO	4	0
EXO	1	0
EXO	2	1
EXO	3	0
EXO	4	0
NAVO	1	2
NAVO	2	0
NAVO	3	0
NAVO	4	0

Figure 22. Experience an Officer Has for Each Job in Years-Manpower Database.

- What is the preference of an officer for a specific job belonging to a specific platform/base?

  The database should provide the means to store the preferences an officer has for specific jobs. The officer's relative preferences for different jobs should be stored too. For example, an officer may prefer to be a Navigation officer for a small ship or better, a Commanding officer for a smaller ship. The figure below provides an example of officers' preferences.

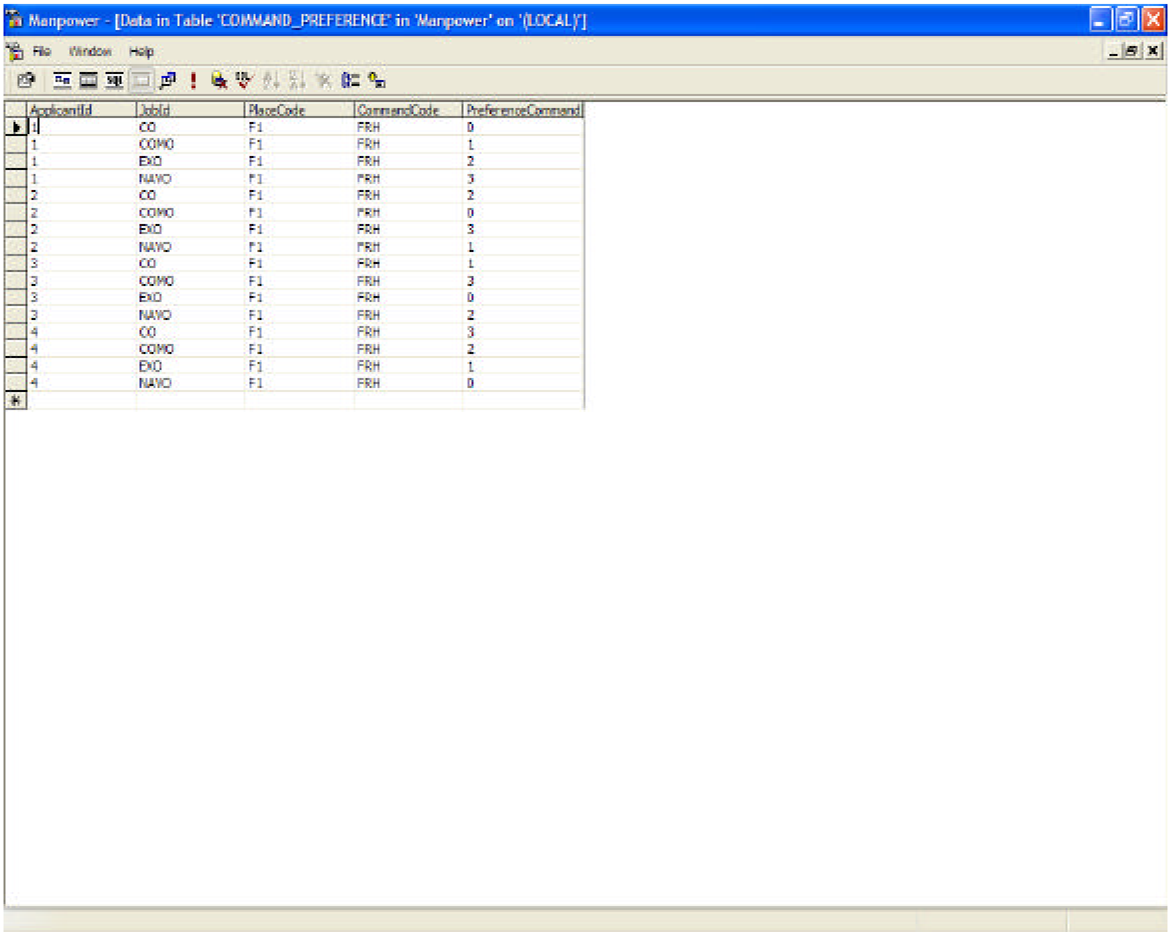


ApplicantId	JobId	PlaceCode	PreferenceApplicant
[redacted]	CO	F1	2
1	COMO	F1	1
1	EXO	F1	3
1	NAVO	F1	0
2	CO	F1	1
2	COMO	F1	2
2	EXO	F1	0
2	NAVO	F1	3
3	CO	F1	3
3	COMO	F1	0
3	EXO	F1	2
3	NAVO	F1	1
4	CO	F1	0
4	COMO	F1	1
4	EXO	F1	2
4	NAVO	F1	3

Figure 23. Officers' Preferences-Manpower Database.

- What are the command's preferences of the officers for a job that belongs under that command?

  The database should also store the various preferences a command has for the officers that may occupy a job under that command. For example the Frigate's Command may prefer to have officer O_1 for the Commanding Officer's position of the FG HYDRA over officer O_2. Figure 22 below provides an example of a commands' preferences.



ApplicantId	JobId	PlaceCode	CommandCode	PreferenceCommand
1	CO	F1	FRH	0
1	COMO	F1	FRH	1
1	EXO	F1	FRH	2
1	NAVO	F1	FRH	3
2	CO	F1	FRH	2
2	COMO	F1	FRH	0
2	EXO	F1	FRH	3
2	NAVO	F1	FRH	1
3	CO	F1	FRH	1
3	COMO	F1	FRH	3
3	EXO	F1	FRH	0
3	NAVO	F1	FRH	2
4	CO	F1	FRH	3
4	COMO	F1	FRH	2
4	EXO	F1	FRH	1
4	NAVO	F1	FRH	0

Figure 24. Commands' Preferences-Manpower Database.

B. ENTITY RELATIONSHIP DIAGRAM

The Entity Relationship Diagram (ERD) is a method of describing the entities and the relationships between them. Since the ERD in this application is quite large, it is reasonable to break it into parts in order to better understand the entities and the relations between them. The entire ERD is presented in the Appendices.

In order to be consistent with the Greek Navy's manpower database requirements as outlined in the previous section and before describing the ERD in depth, we provide a table listing all the entities with a short description of each.

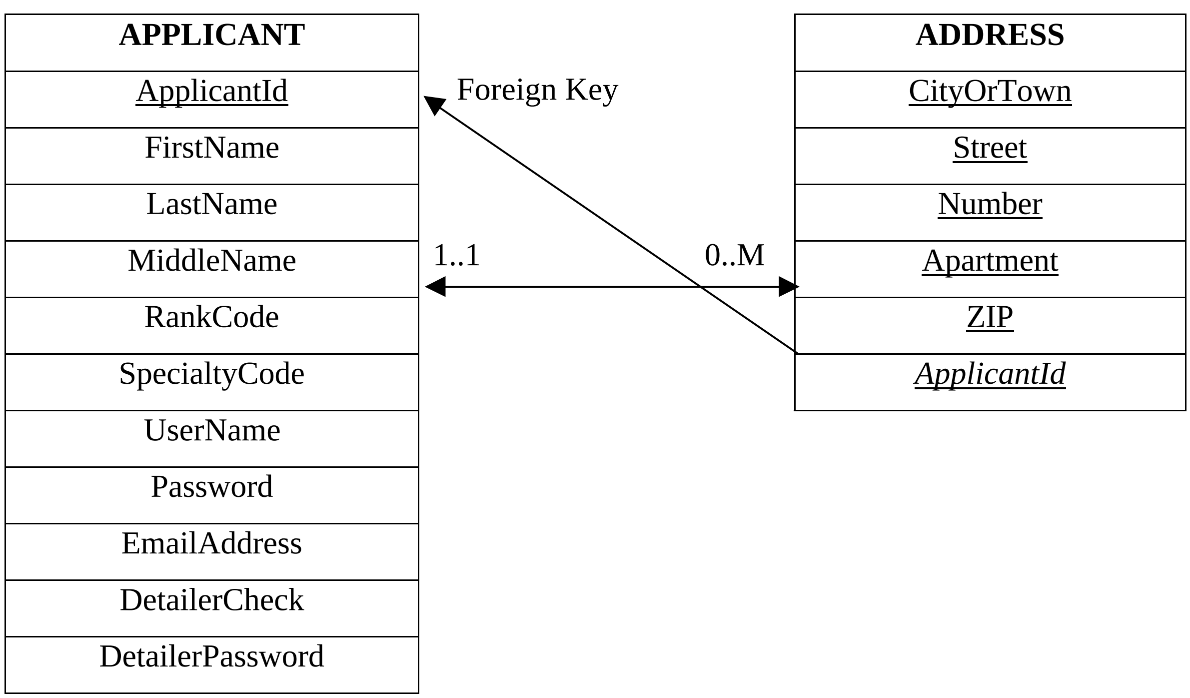
Number	Entities	Description
1	JOB	Includes all the job information such as the job name and the experience required for a job. It also contains the priority of a job. The priority is ranked along ascending importance in levels 1 through 10. It is stored by the detailer and describes the importance of a particular job.
2	APPLICANT	It contains the officer's information like the officer's identification number, the officer's last name, first name, middle name, email address, username and password for the website and finally the officer's rank and specialty.
3	ADDRESS	It includes the officer's address information, like the city that the officer lives in, the street name and number, the apartment and the zip code.
4	PHONE	It includes the officer's phone numbers, like the home phone number, the cell phone number or any additional phone number the officer might have.
5	COMMAND	It includes the command's data like the command's name and the username and password that is used for the website.
6	PLACE	It includes information like the base's/platform's name (where different kinds of jobs exist) and image (a photo of the base/platform).
7	ASSIGNMENT	It includes all assignment information like the job, the platform/base and the officer that is assigned a particular job, the report date and the detach date.
8	RANK	It includes all the possible ranks that an officer may have or that a job requires from an officer to have.
9	LANGUAGE	It includes all the possible languages that an officer may speak or that a job requires from an officer to speak.
10	SPECIALTY	It includes all the possible specialties that an officer may have or that a job requires from an officer to have.
11	QUALIFICATION	It includes all the possible qualifications that an officer may have or that a job requires from an officer to have. An example of it is an entire catalog of all the schools or educational programs.
12	CREDENTIALS	It includes all the possible credentials that an officer may have or that a job requires from an officer to have. Some of them are diligence,

Number	Entities	Description
		discretion, secrecy, discipline, etc.
13	EXPERIENCE	It includes the experience in years that an officer has for a particular job. For example an officer y has 1 year of experience for the job x. This experience can be directly compared with the experience that a job requires, which is stored in the table of the entity JOB.
14	APPLICANT PREFERENCE	It describes an officer's preference for a particular job. It includes the officer, the job, the platform/base and the preference. The latter one is ranked along descending importance in levels 1 through 10.
15	COMMAND PREFERENCE	It describes a command's preference concerning a particular job that belongs to this command, for which officer the command prefers to occupy that job. It includes the officer, the job, the platform/base, the command and the preference. The latter one is ranked along descending importance in levels 1 through 10.

Table 5. All Entities with a Short Description of Each.

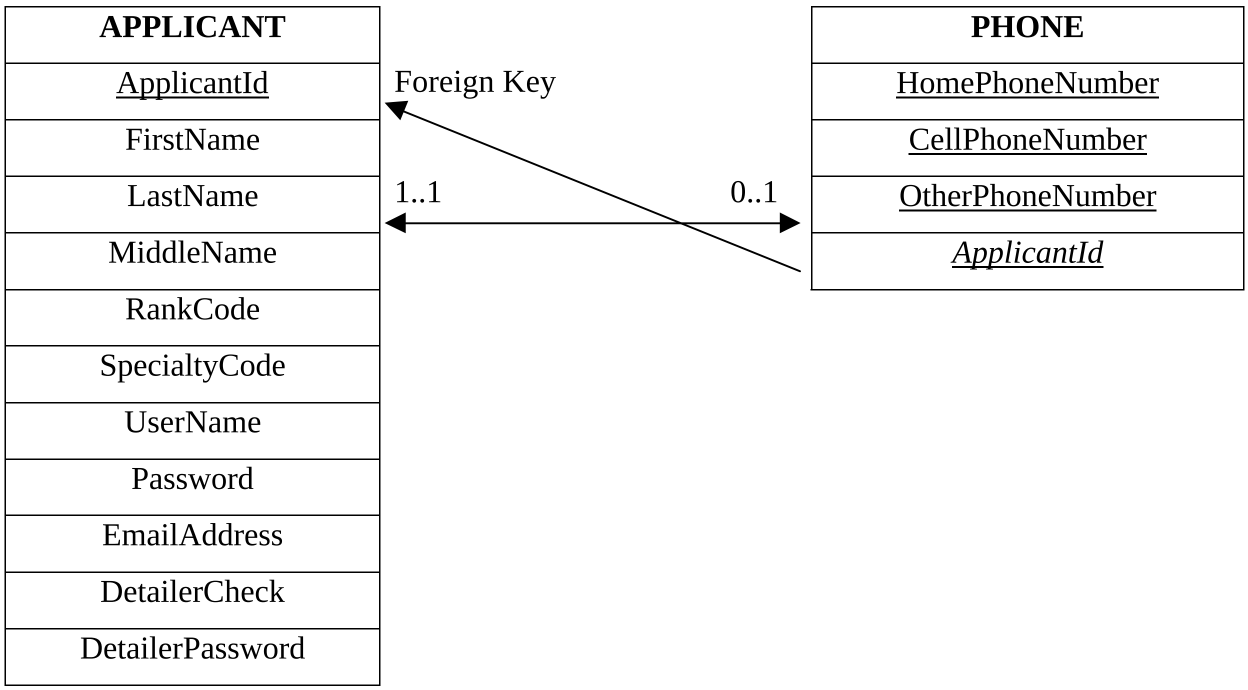
Below we present the various segments of the Manpower Database ERD.

1. Applicant-Address



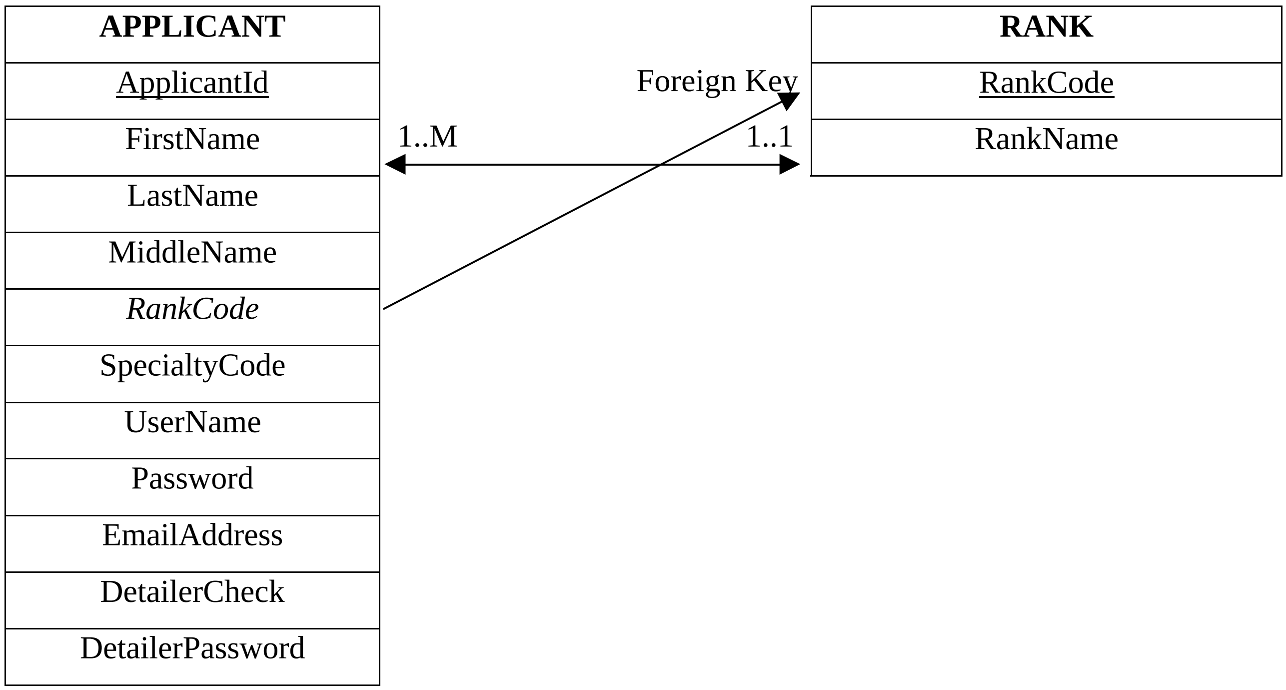
The relation is one-to-many since an officer may live in more than one residence. Thus, the officer may have more than one address. The attribute ApplicantId is the foreign key from the entity ADDRESS referencing the entity APPLICANT.

2. Applicant-Phone



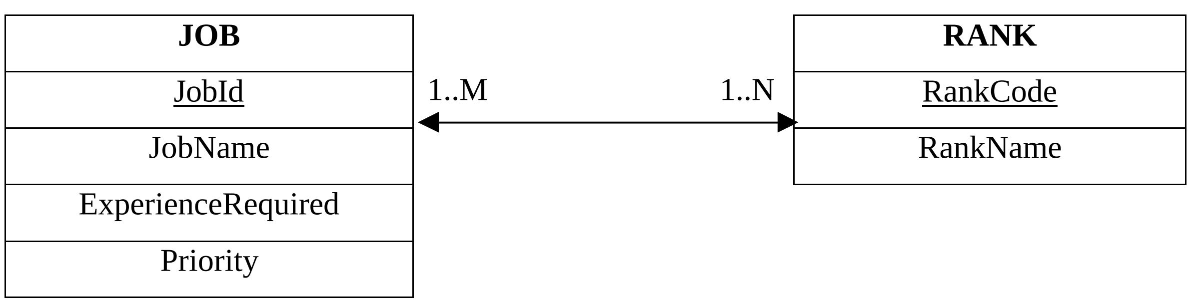
The relation is one-to-one. An officer may have one home phone number or one cellular phone number or possibly another phone number. The attribute ApplicantId is the foreign key from the entity PHONE referencing the entity APPLICANT.

3. Applicant-Rank



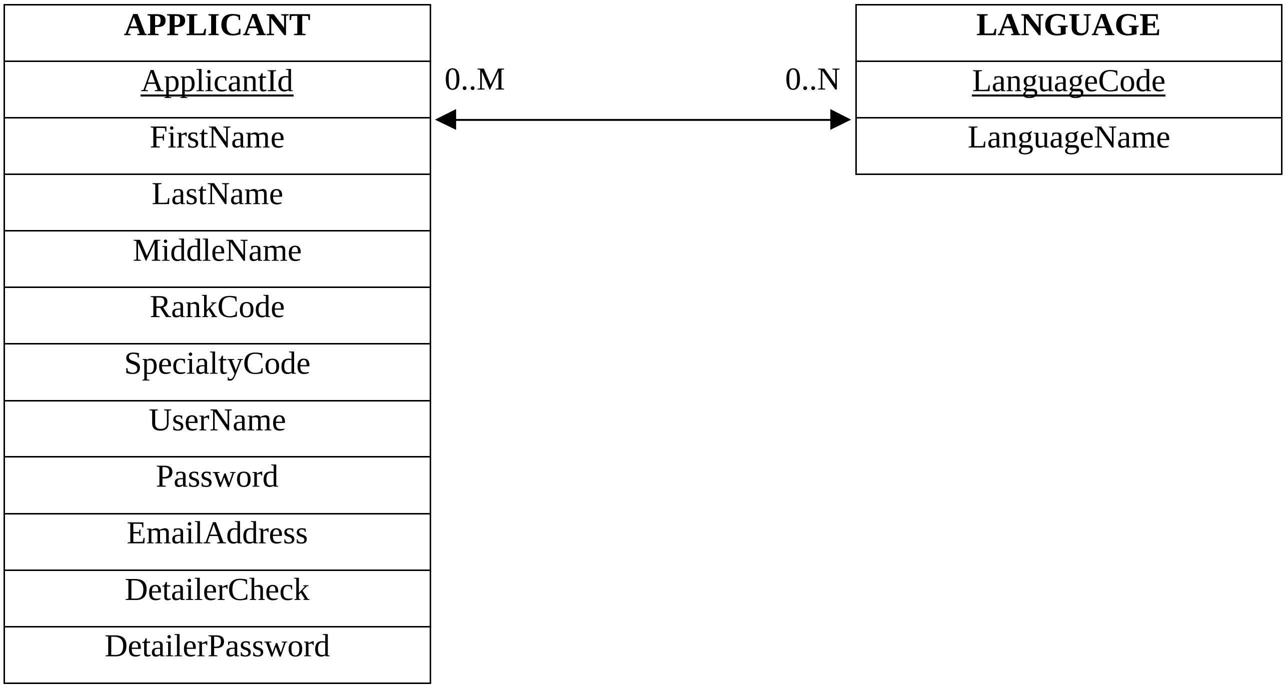
The relation is many-to-one since an officer has only one rank, but a rank may be applied to many officers. For example an officer can have only the rank O2, but O2 can be the rank of more officers. The attribute RankCode is the foreign key from the entity APPLICANT referencing the entity RANK.

4. Job-Rank



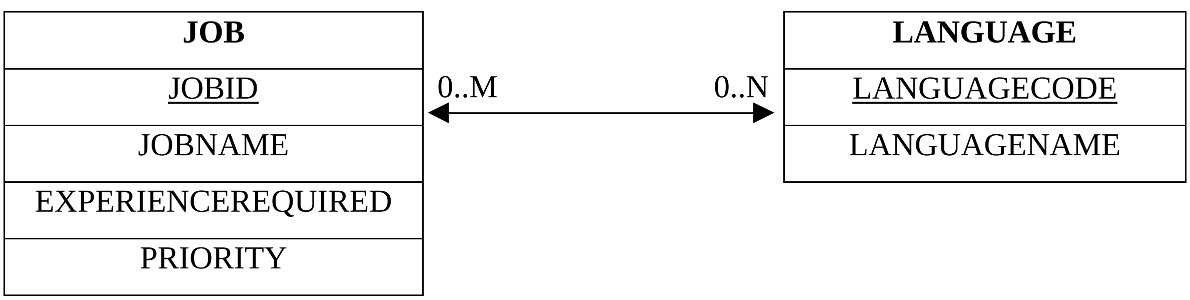
The relation is many-to-many since the ranks that a job requires for the officers to have may be more than one. Also a rank may be required for more than one job. For example a Commander can be an officer with rank O3 or O4 or O5, and an officer with rank O4 can be a Commander or a Base Commander.

5. Applicant-Language



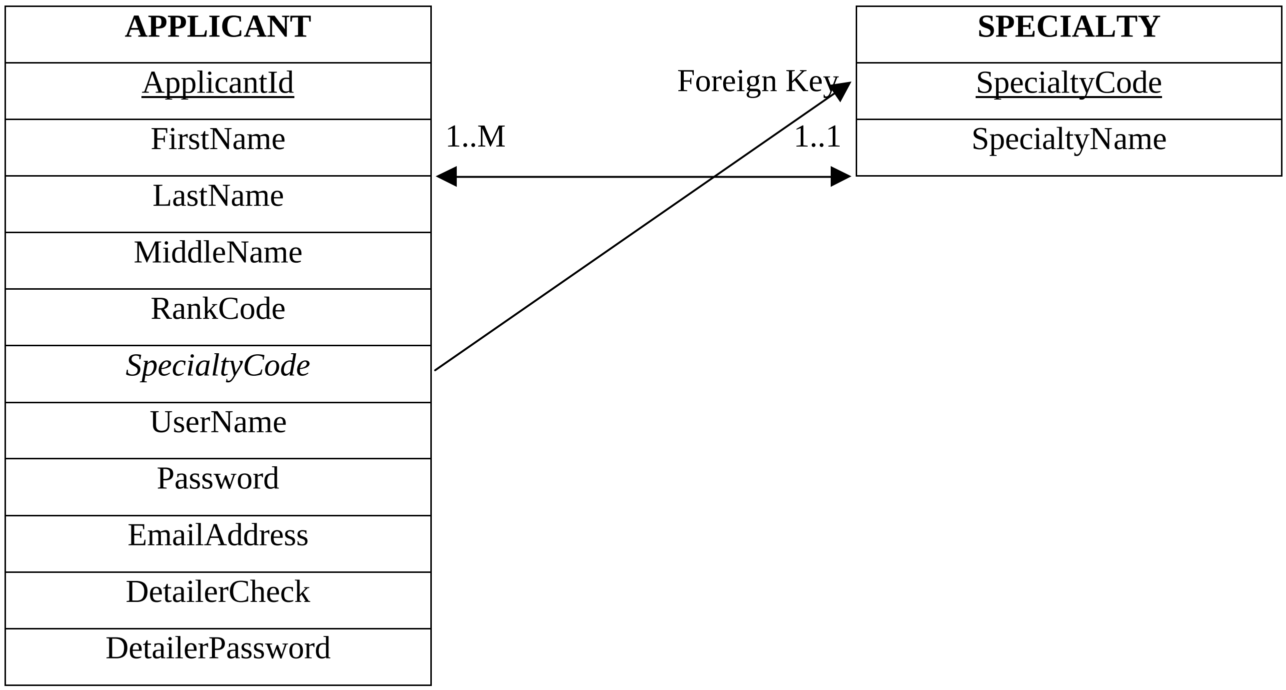
The relation is many-to-many since an officer can speak many languages, and since a language can be spoken by many officers. For example an officer can speak English and German, but also the German language can be spoken by many officers.

6. Job-Language



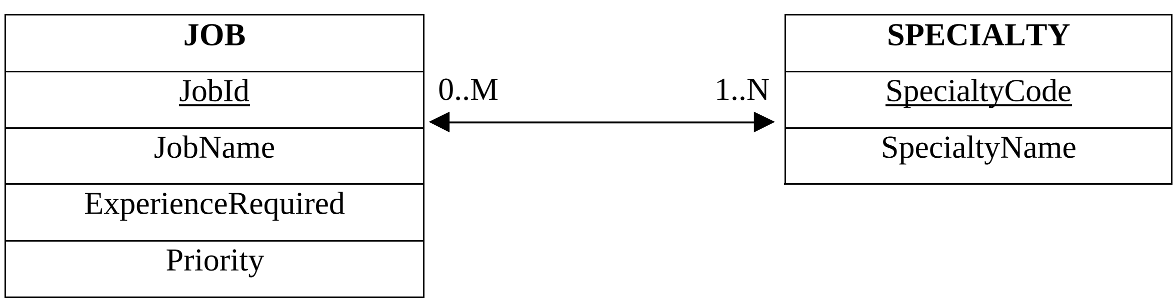
The relation is many-to-many since there can be many languages that a job requires for the officers to speak. Also a language may be a requirement for many jobs. For example a job can require an officer to speak both English and Spanish, while English can be considered by many jobs as a requirement.

7. Applicant-Specialty



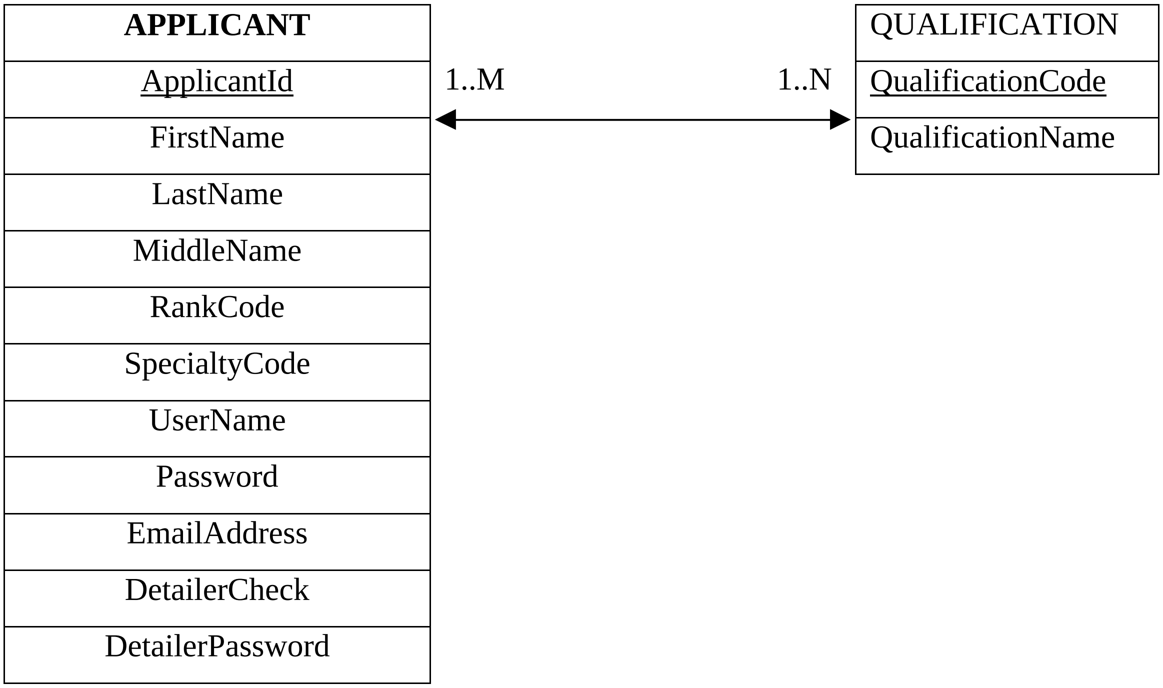
The relation is many-to-one since an officer has only one specialty, but a specialty may be applied to many officers. For example, an officer can only have one specialty like the Weapons specialty. The Weapon specialty can be assigned to many officers. The attribute SpecialtyCode is the foreign key from the entity APPLICANT referencing the entity SPECIALTY.

8. Job-Specialty



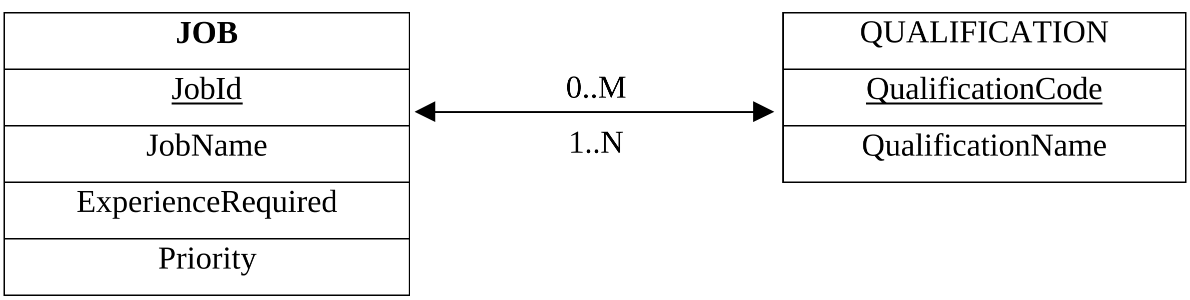
The relation is many-to-many since the specialties that a job requires for the officers to have may be more than one. Also a specialty may be applied for more than one job. For example, a Commander can be an officer with Weapons specialty, or an officer with Navigation specialty, while the Weapons specialty can be a requirement for both the Commander and the Weapons officer.

9. Applicant-Qualification



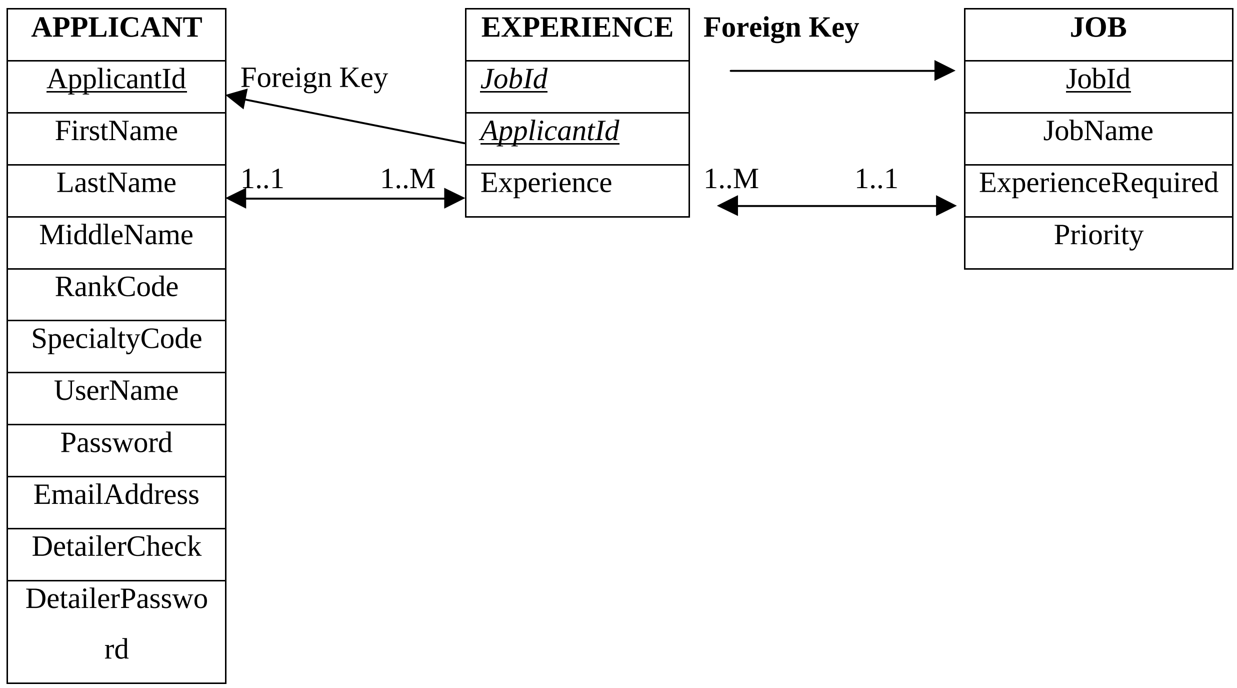
The relation is many-to-many since an officer can have many qualifications, and one qualification can be applied to many officers. For example, an officer can be a graduate of both the Greek and the US Weapons Schools. Also, there could be many officers that graduated the Greek Weapons School.

10. Job-Qualification



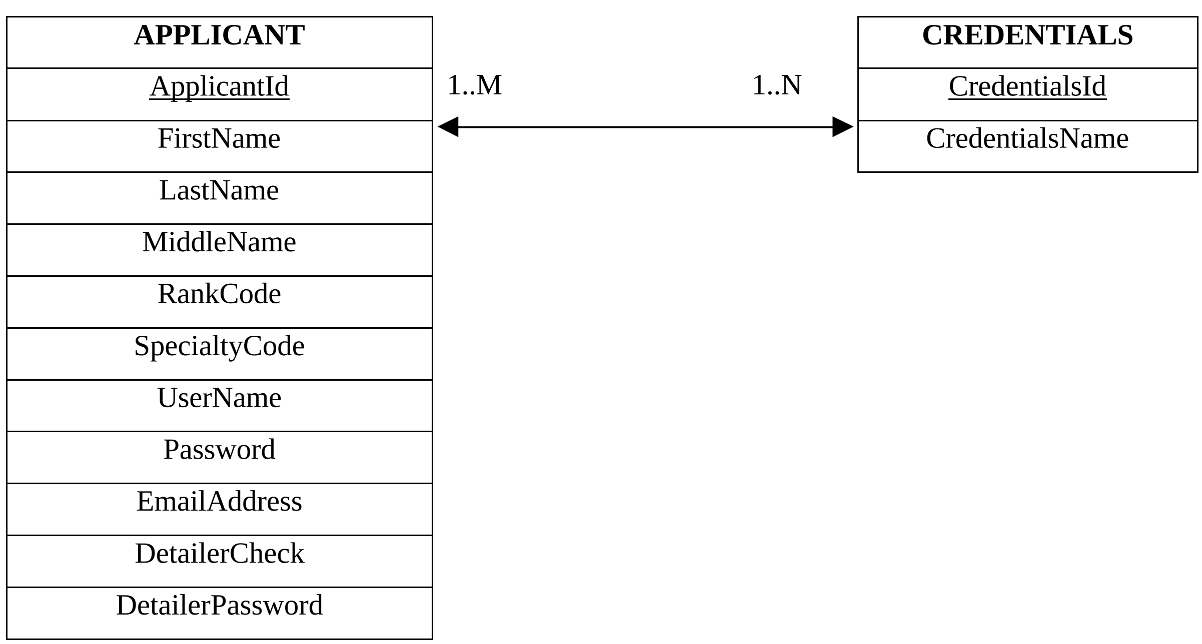
The relation is many-to-many since a job can have many qualifications, and one qualification can be applied to many jobs. For example a job may require that the officers should have been graduated from both the Greek and the US Weapons Schools and the Greek Weapons School could be a requirement for many jobs.

11. Applicant-Experience-Job



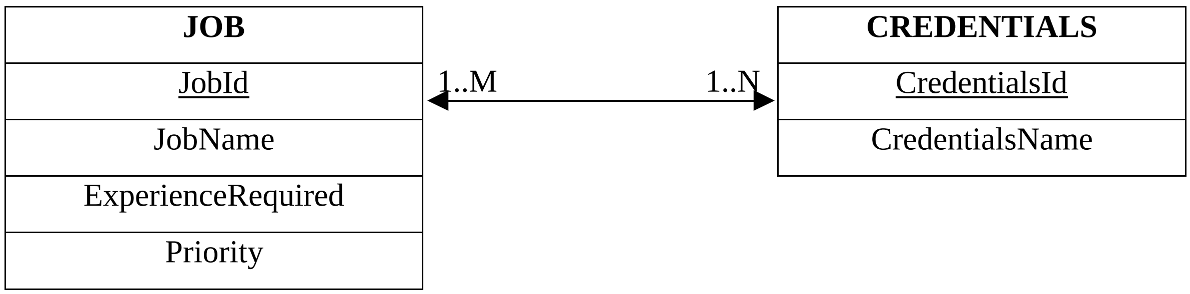
These are the relations between the three entities, the APPLICANT, the EXPERIENCE, and the JOB entity. The attribute ApplicantId is the foreign key from the entity EXPERIENCE referencing the entity APPLICANT. The attribute JobId is the foreign key from the entity EXPERIENCE referencing the entity JOB.

12. Applicant-Credentials



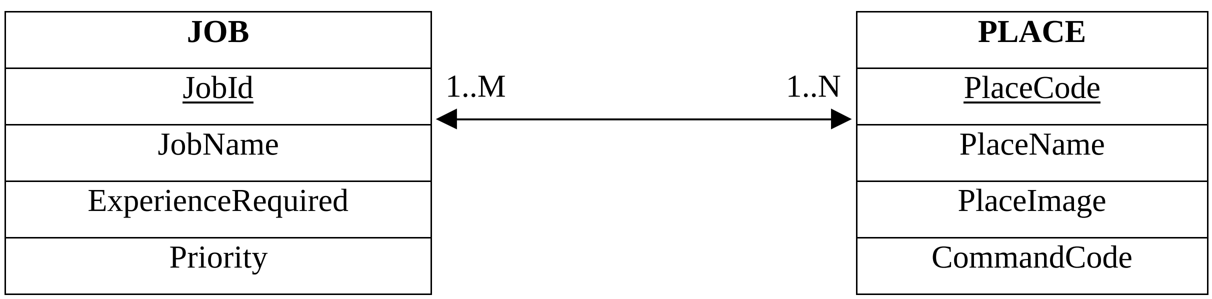
The relation is many-to-many since an officer can have many credentials, and a credential can be assigned by many officers. For example an officer can be diligent and brave, but also bravery can be a credential for many officers.

13. Job-Credentials



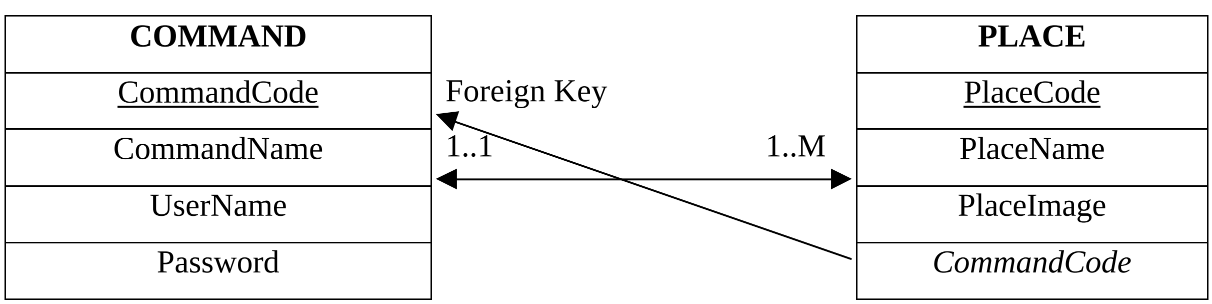
The relation is many-to-many since a job may require many credentials, and one credential can be applied to many jobs. For example a job may require that the officers should be diligent and brave and also bravery could be a requirement for many jobs.

14. Job-Place



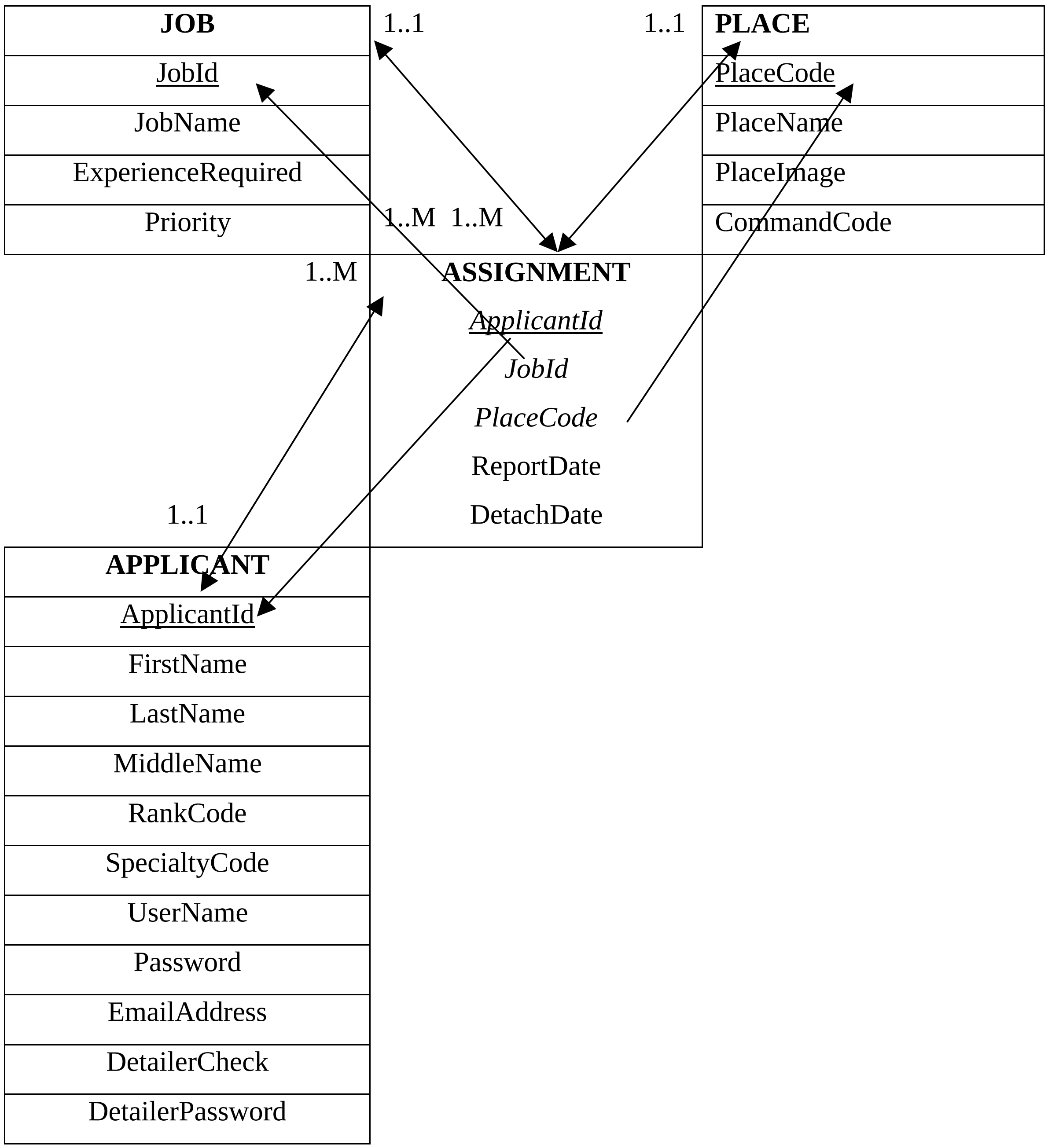
The relation is many-to-many since a platform/base may have more than one job. Also, a job can be in more than one platform/base. For example, the Navigation job is a job in every ship. Also a ship has many jobs like the navigation and the weapons jobs.

15. Command-Place



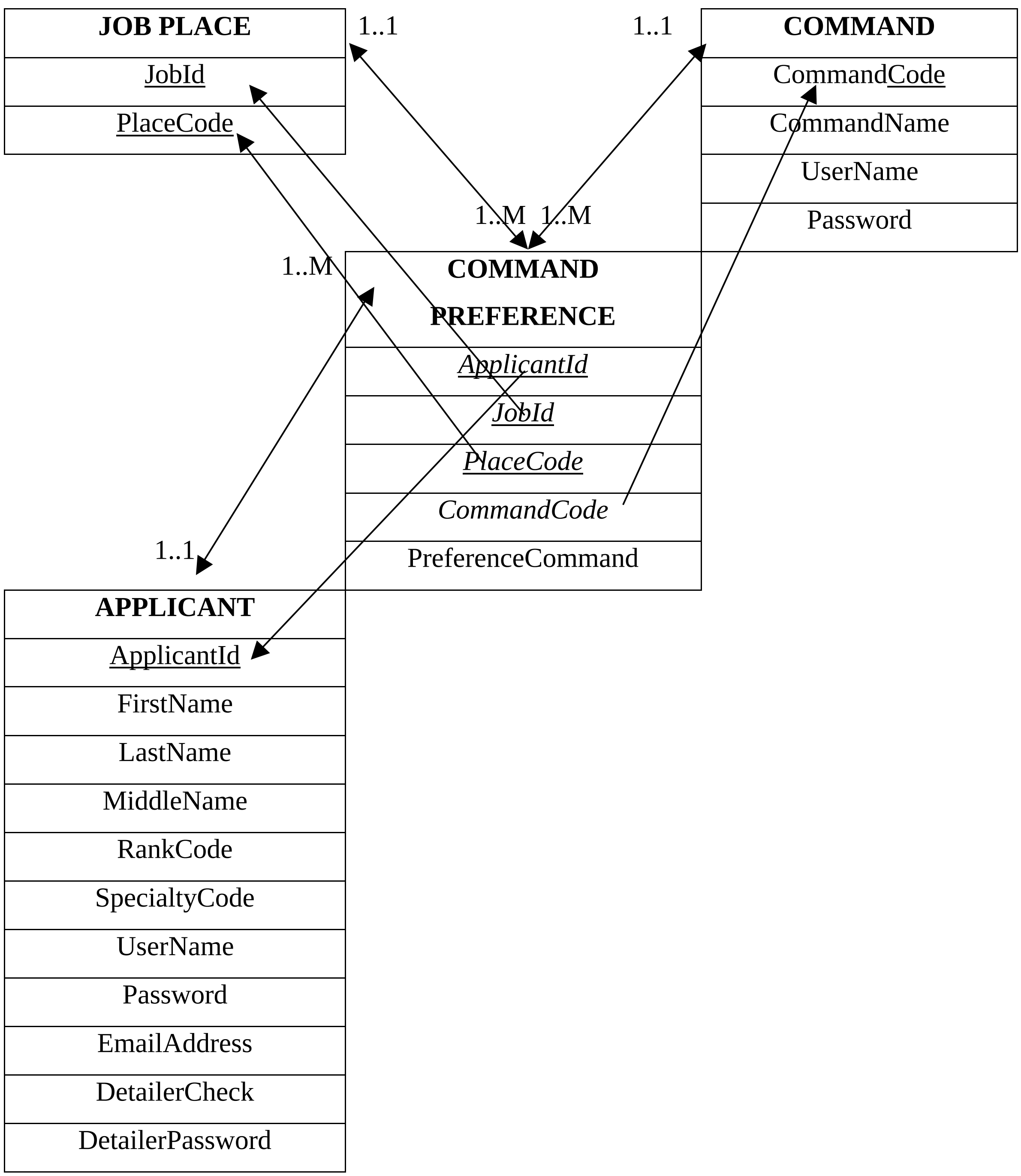
The relation is one-to-many. A Command may have many Platforms/Bases under its command. The Platform/Base belongs to only one Command. For example, the Frigates Headquarters have many ships under their command (e.g. FG HYDRA, FG SPETSAI). On the other hand, FG HYDRA belongs only to the Frigates Headquarters.

16. Assignment-Job-Place-Applicant



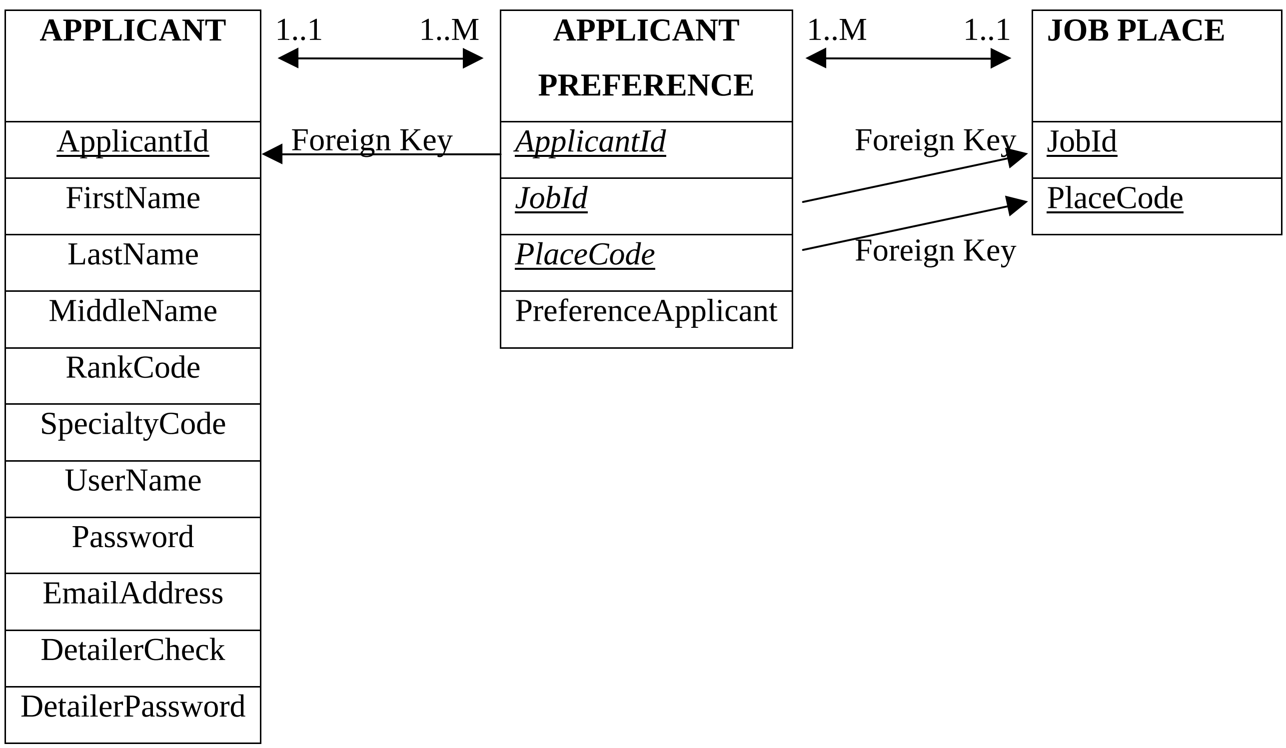
This is a ternary relationship between the ASSIGNMENT, JOB, PLACE, APPLICANT entities. The attribute ApplicantId is the foreign key from the entity ASSIGNMENT referencing the entity APPLICANT. The attribute JobId is the foreign key from the entity ASSIGNMENT referencing the entity JOB. The attribute PlaceCode is the foreign key from the entity ASSIGNMENT referencing the entity PLACE.

17. Command Preference-Command- Job Place-Applicant



This is a ternary relationship between the COMMAND PREFERENCE, COMMAND, JOB PLACE, APPLICANT entities. The attribute ApplicantId is the foreign key from the entity COMMAND PREFERENCE referencing the entity APPLICANT. The attribute JobId is the foreign key from the entity COMMAND PREFERENCE referencing the entity JOB PLACE. The attribute PlaceCode is the foreign key from the entity COMMAND PREFERENCE referencing the entity JOB PLACE. The attribute CommandCode is the foreign key from the entity COMMAND PREFERENCE referencing the entity COMMAND.

18. Applicant Preference- Job Place-Applicant



This is a ternary relationship between the APPLICANT PREFERENCE, JOB PLACE, APPLICANT entities. The attribute ApplicantId is the foreign key from the entity APPLICANT PREFERENCE referencing the entity APPLICANT. The attribute JobId is the foreign key from the entity APPLICANT PREFERENCE referencing the entity JOB PLACE. The attribute PlaceCode is the foreign key from the entity APPLICANT PREFERENCE referencing the entity JOB PLACE.

C. RELATIONAL MODEL

The ERD can be automatically transformed into a set of tables which form a schema in a target database management system such as SQL Server 2000 or Microsoft Access. The attributes that are underlined below are the <u>primary keys</u>, the values of which uniquely identify each row of the corresponding table. The attributes that in italics are foreign keys, which are the primary keys of other tables embedded in order to represent a relationship between the two tables.

APPLICANT (<u>ApplicantId</u>, FirstName, LastName, MiddleName, *RankCode_FK*, *SpecialtyCode_FK*, UserName, Password, EmailAddress, DetailerCheck, DetailerPassword)

ApplicantId is the officer's identification number (e.g. A001), FirstName is the officer's first name (e.g. Kyriakos), LastName is the officer's last name (e.g. Sergis), MiddleName is the officer's middle name (e.g. Nikitas), UserName and Password are the officer's user name and password the officer uses for the web site, EmailAddress is the officer's email address, DetailerCheck is a special Boolean attribute that is 'yes' for the detailer and 'no' for the rest officers, and DetailerPassword is an extra password that only the detailer has.

JOB (<u>JobId</u>, JobName, ExperienceRequired, Priority)

JobId is the job's identification number (e.g. BCO), JobName is the job's name (e.g. Base Commander), and Priority is the priority of the job as it was described previously (e.g. 9).

ADDRESS (<u>CityOrTown</u>, <u>Street</u>, <u>Number</u>, <u>Apartment</u>, <u>ZIP</u>, <u>*ApplicantId*</u> *FK*)

CityOrTown is the city or town the officer lives (e.g. Athens), Street is the street the officer's residence exists (e.g. Markora), Number is the number of the building the officer's residence sits (e.g. 302), Apartment is the number of the officer's apartment (e.g. A), and ZIP is the ZIP or Postal Code of the area the officer lives.

PHONE (<u>*ApplicantId*</u> *FK*, <u>HomePhoneNumber</u>, <u>CellPhoneNumber</u>, <u>OtherPhoneNumber</u>)

HomePhoneNumber is the officer's home phone number, CellPhoneNumber is the officer's cellular phone number, and OtherPhoneNumber is any other phone number the officer has.

RANK (<u>RankCode</u>, RankName)

RankCode is the rank code (e.g. O3), and RankName is the name of the rank (e.g. Lieutenant)

LANGUAGE (<u>LanguageCode</u>, LanguageName)

LanguageCode is the language code (e.g. EN), and LanguageName is the name of the language (e.g. English)

SPECIALTY (<u>SpecialtyCode</u>, SpecialtyName)

SpecialtyCode is the specialty code (e.g. WPS), and SpecialtyName is the name of the specialty (e.g. Weapons)

QUALIFICATION (<u>QualificationCode</u>, QualificationName)

QualificationCode is the qualification code (e.g. WPSGR), and QualificationName is the name of the qualification (e.g. Weapons School Greece)

EXPERIENCE (<u>*JobId*</u> *FK*, <u>*ApplicantId*</u> *FK*, Experience)

Experience is the years of experience e.g. 3 that the officer with Identification Number (ID) ApplicantId has for the job with ID JobId.

COMMAND (CommandCode, CommandName, UserName, Password)

CommandCode is the command code (e.g. FRH), CommandName is the name of the command (e.g. Frigates Headquarters), and UserName, Password are special user names and passwords for each one of the commands.

PLACE (PlaceCode, PlaceName, PlaceImage, *CommandCode FK*)

PlaceCode is the Platform or Base code (e.g. F-450), PlaceName is the name of the Base/Platform (e.g. FG HYDRA), and PlaceImage is the image of the Platform/Base (e.g. F-450.jpeg)

APPLICANT PREFERENCE (*JobId FK*, *ApplicantId FK*, *PlaceCode FK*, PreferenceApplicant)

PreferenceApplicant is the preference (e.g. 7) of the officer with ID ApplicantId for the job with ID JobId that is cited in the Platform/Base with code PlaceCode.

COMMAND PREFERENCE (JobId FK, ApplicantId FK, PlaceCode FK, CommandCode FK, PreferenceCommand)

PreferenceCommand is the preference (e.g. 7) of the command with command code CommandCode for the officer with ID ApplicantId for the job with ID JobId that is sited in the Platform/Base with code PlaceCode.

CREDENTIALS (CredentialsId, CredentialsName)

CredentialsId is the ID of the credential (e.g. 001), and CredentialsName is the name of the credential (e.g. diligence)

ASSIGNMENT (*ApplicantId FK*, *JobId FK*, *PlaceCode FK*, ReportDate, DetachDate)

ReportDate and DetachDate are the report and detach dates of each one of the assignments. Each assignment has also the ApplicantId of the officer who is assigned the job with ID JobId that sites in the Base/Platform with code PlaceCode.

In order to achieve redundancy of tables and to perform some additional functionality, the following tables/entities are also defined.

APPLICANT CREDENTIALS (*ApplicantId FK*, *CredentialsId FK*, CredentialsGrade)

CredentialsGrade is the grade (e.g. 7) of the credential with ID CredentialsId that an officer with ID ApplicantId has.

APPLICANT LANGUAGE (*ApplicantId FK*, *LanguageCode FK*, LanguageDegree)

LanguageDegree is the grade (e.g. 70) of the language with code LanguageCode that an officer with ID ApplicantId has.

JOB CREDENTIALS (*JobId FK*, *CredentialsId FK*, CredentialsGrade)

CredentialsGrade is the minimum grade (e.g. 8) of the credential with ID CredentialsId that an officer should have to be qualified for the job with ID JobId.

JOB LANGUAGE (*JobId FK*, *LanguageCode FK*, LanguageDegree)

LanguageDegree is the minimum grade (e.g. 8) of the language with code LanguageCode that an officer should have to be qualified for the job with ID JobId.

JOB PLACE (*JobId FK*, *PlaceCode FK*)

JobId is the ID of the job and PlaceCode refers to the Platform/Base the job belongs to.

JOB QUALIFICATION (*JobId FK*, *QualificationCode FK*)

JobId is the ID of the job and QualificationCode refers to the qualification that is required for that job.

JOB RANK (JobId FK, RankCode FK)

JobId is the ID of the job and RankCode refers to the rank that this job requires from an officer to have.

JOB SPECIALTY (JobId FK, SpecialtyCode FK)

JobId is the ID of the job and SpecialtyCode refers to the specialty that this job requires from an officer to have.

QUALIFICATION APPLICANT (ApplicantId FK, QualificationCode FK)

ApplicantId is the ID of the officer and QualificationCode refers to the qualification that this officer has.

The table schema described above is actually the set of tables that was entered into SQL Server 2000 in the Manpower Database and is described in section B.

The Manpower database meets all the requirements that are necessary for the distribution of officers to jobs. The following chapter makes one step further on this direction. It describes the algorithm, which is responsible for creating that distribution. Then the detailer can intervene and change that distribution according to the Navy's needs.

IV. DECISION MODEL

In Chapter III, we discussed the design of the database that holds all the relevant information for the officers to jobs distribution. This distribution is achieved by an algorithm that, when executed, solves the multi-criteria problem. The detailer can alter any part of the entire solution according to the wishes of the Navy, and subsequently see what effect it has on the overall "goodness" of the assignment.

This chapter presents in full detail the philosophy and implementation of the algorithm and utility function. In this chapter and in order to simplify the algorithm, the word "job" will refer to the combination of a job with a specific platform/base.

A. DECISION VARIABLES

In order for the algorithm to determine the most suitable officer for a specific job, the algorithm takes into account the following decision variables.

- Rank
- Specialty
- Qualifications
- Language
- Credentials
- Experience
- Officer's Preference
- Command's Preference

These variables are expressed in the form of values, which determine the suitability of an officer for a specific job. This suitability is named H_{ij} (where i, j are the indices of the i-th job J_i and j-th officer O_j accordingly) and is expressed as a function of the above eight variables.

H_{ij} = Function (Rank, Specialty, Qualifications, Language, Credentials, Experience, Officer's Preference, Command's Preference)

More specifically, the values of each one of the decision variables are as follows.

1. Rank

The Rank is expressed by a value, which is 1 if the O_j officer has the appropriate rank for the J_i job or 0 if the officer has not.

2. Specialty

The Specialty is expressed by a value, which is 1 if the O_j officer has the appropriate specialty for the J_i job or 0 if the officer has not.

3. Qualifications

The Qualifications are expressed by a value, which is 1 if the O_j officer has the appropriate qualifications (qualification is considered the education of the officer for a specific job) for the J_i job or 0 if the officer has not.

4. Language

The Language is expressed by a value in the real interval [0,10], computed as follows: First, the summation of the grades the O_j officer has for the languages that are required for the J_i job is computed. Then the summation of the minimum grades of these languages required for the J_i job is computed also. If the first summation is smaller than the second, the Language variable's value is 0. Else, the Language variable's value is a number between 1 and 10, according to a formula that takes into account the relative difference of these two summations. Below is pseudocode that describes the computation of the Language variable's value. Keep in mind that the maximum grade of each language is 200.

Step 1: SUM1 = (Sum of grades the O_j officer has for the languages required for J_i job)

Step 2: SUM2 = (Sum of minimum grades of the languages required for J_i job)

Step 3: COUNT = (Number of the languages required for J_i job)

Step 4: IF (SUM1 < SUM2) THEN

BEGIN

Step 5: $Language_{ij}$ -> 0

END

Step 6: ELSE

BEGIN

Step 7: IF (COUNT x 200 = SUM2)

BEGIN

Step 8: $Language_{ij}$ -> 1

END

Step 9: ELSE

BEGIN

Step 10: $Language_{ij}$ -> [(SUM1-SUM2) x 9 / ((COUNT x 200)-SUM2)] + 1

END

END

The following example makes it clear.

Consider an officer O_1 that is eligible for a job J_1. J_1 job requires the languages English and German with minimum grades 160 and 120 (0 is the minimum and 200 is the maximum grade) accordingly. Officer O_1 speaks English with a grade of 180 and German with a grade of 110. The value of the variable Language for the O_1 officer and J_1 job is as follows.

Step 1: SUM1 = 180 + 110 = 290

Step 2: SUM2 = 160 + 120 = 280

Step 3: COUNT = 2

Step 6: SUM1 = 290 > 280 = SUM2

Step 9: COUNT x 200 = 400 > 280 = SUM2

SUM1 – SUM2 = 10

COUNT x 200 – SUM2 = 400 – 280 = 120

Step 10: $Language_{11}$ = [10 x 9 / 120] + 1 = 1.75

5. Credentials

The Credentials variable is an integer in the interval [0,10] and is computed as follows: First, the summation of the grades the O_j officer is evaluated for the credentials that are required for the J_i job is computed. Then the summation of the minimum grades of these credentials required for the J_i job is computed too. If the first summation is smaller than the second, the Credentials variable's value is 0. Else, the Credentials variable's value is a number between 1 and 10, according to a formula that takes into account the relative difference of these two summations. Below is a pseudocode that describes the computation of the Credentials variable's value. Have in mind that the maximum grade of each language is 10.

```
Step 1:     SUM1 = (Sum of grades the O_j officer is evaluated for the credentials
            required for J_i job)
Step 2:     SUM2 = = (Sum of minimum grades of the credentials required for J_i job)
Step 3:     COUNT = (Number of the credentials required for J_i job)
Step 4:     IF (SUM1 < SUM2) THEN
            BEGIN
Step 5:         Credentials_ij -> 0
            END
Step 6:     ELSE
            BEGIN
Step 7:         IF (COUNT x 10 = SUM2)
                BEGIN
Step 8:             Credentials_ij -> 1
                END
Step 9:         ELSE
                BEGIN
Step 10:            Credentials_ij -> [(SUM1-SUM2) x 9 / ((COUNT x 10)-
                    SUM2)] + 1
                END
            END
```

The following example makes it clear. Consider again officer O_1 that is eligible for the job J_1. J_1 job requires the credentials Diligence and Bravery with minimum grades 9 and 8 (0 is the minimum and 10 is the maximum grade) accordingly. O_1 officer's credential grades are 10 and 8 for Diligence and Bravery accordingly. The value of the variable Credentials for the O_1 officer and J_1 job is as follows.

Step 1: SUM1 = 10 + 8 = 18

Step 2: SUM2 = 9 + 8 = 17

Step 3: COUNT = 2

Step 6: SUM1 = 18 > 17 = SUM2

Step 9: COUNT x 10 = 20 > 17 = SUM2

SUM1 – SUM2 = 1

COUNT x 10 – SUM2 = 20 – 17 = 3

Step 10: Credentials_{11} = [1 x 9 / 3] + 1 = 4

6. Experience

The Experience variable is expressed by a value in the real interval [0,10], computed as follows. If the experience the O_j officer has on the J_i job is smaller than the minimum experience required for the J_i job, the Experience variable's value is 0. Else, the Experience variable's value is a number between 1 and 10, according to a formula that takes into account the relative difference of the experience the O_j officer has on the J_i job and the minimum experience required for the J_i job. Below is pseudocode that describes the computation of the Experience variable's value. Keep in mind that the maximum experience an officer can have for a job is 15 years, and the minimum experience required for a job cannot be more than 10 years.

Step 1: OfficerExperience = (Experience the O_j officer has on the J_i job)

Step 2: JobExperience = (Minimum experience required for J_i job)

Step 3: IF (OfficerExperience < JobExperience) THEN

BEGIN

Step 4: Experience_{ij} -> 0

END

Step 5: ELSE

BEGIN

Step 6: Experience_{ij} -> [(OfficerExperience – JobExperience) x 9 / (15 – JobExperience)] + 1

END

The example with the same job and officer makes it clear. Consider again officer O_1 that is eligible for the job J_1. J_1 job requires 3 years of experience. If the O_1 officer has 1 year of experience on that job, the value of the Experience variable is 0. If the O_1 officer has 4 years of experience on that job, the value of the Experience variable is [(4 – 3) x 9 / (15 – 3)] + 1 on a total of 1.75.

7. Officer's Preference

The Officer's Preference value is an integer in the interval [1,10]. Since the value stored in the APPLICANT PREFERENCE table is ranked by descending importance in levels 1 through 10, the Officer's Preference value is 11 minus the APPLICANT PREFERENCE table value. If the officer does not have any preference for the job, the Officer's Preference value is 0.

8. Command's Preference

The Command's Preference value is an integer in the interval[1,10]. Since the value stored in the COMMAND PREFERENCE table is ranked by descending importance in levels 1 through 10, the Command's Preference value is 11 minus the COMMAND PREFERENCE table value. If the command does not have any preference for the officer occupying the job that belongs to that command, the Command's Preference value is 0.

9. Computation of the Goodness of Fit Index, H_{ij}

If the O_j officer has a value of 0 for any of the Rank, Specialty or Qualifications variables concerning job J_i, the H_{ij} value is NULL. This means that the O_j officer is not eligible for the job J_i.

In the case that O_j officer is eligible for the J_i job, the H_{ij} value is a function of the remaining five decision variables. Each one of these variables may have different importance, measured by the coefficient that is stored in the COEFFICIENT table (a table that contains the coefficients and the coefficient numbers that are used to weight the importance of each criterion described above). It is actually a weight factor that, when multiplied by the corresponding variables value, gives a weighted estimation of the variables' importance.

$$H_{ij} = 1 + \sum_{k=1}^{k=5} \left(C_k \text{ x Variable}_{ij} \right)$$

Addition with number 1 is necessary since the summation can be a non-negative number and 0 values are not desirable for the utility function as we shall see below. c_k is the decision variable coefficient's value.

Now it may be seen why it is important normalize all the variable values to have the same maximum and minimum scores, 10 and 0 respectively. If one variable has a greater maximum value than the rest, it would have a bigger advantage over the remaining variables and conversely, if one variable has lesser minimum value than the rest, it would suffer a bigger disadvantage compared to the remaining variables especially when multiplied by a coefficient.

The following example makes the computation of the H_{ij} function clear.

Consider again officer O_1 and job J_1. If one of the $Rank_{11}$, $Specialty_{11}$, or $Qualifications_{11}$ values is 0, then the O_1 officer is not eligible for the J_1 job, and the H_{11} value is NULL.

$$H_{11} = NULL$$

Assume that $Rank_{11}$, $Specialty_{11}$, or $Qualifications_{11}$ value are all greater than 0 as follows:

- Language Coefficient is 1. $c_1 = 1$
- Credentials Coefficient is 1. $c_2 = 1$
- Experience Coefficient is 1. $c_3 = 1$
- Officer's Preference Coefficient is 2. $c_4 = 2$
- Command's Preference Coefficient is 2. $c_5 = 2$
- J_1 job requires the languages English and German with minimum grades 160 and 120 accordingly. Officer O_1 speaks English with a grade of 180 and German with a grade of 110.
- J_1 job requires the credentials Diligence and Bravery with minimum grades 9 and 8 accordingly. O_1 officer's credential grades are 10 and 8 for Diligence and Bravery accordingly.

- J_1 job requires 3 years of experience. Officer O_1 has 4 years of experience on that job.
- Officer O_1 preference for the J_1 job, as it is stored in the APPLICANT PREFERENCE table is 2.
- There is no preference of the command concerning the J_1 job for the O_1 officer. Thus, there is no record in the COMMAND PREFERENCE table.

The H_{11} value is computed as follows.

- From above, $Language_{11} = 1.75$
- From above, $Credentials_{11} = 4$
- $Experience_{11} = [(4 – 3) \times 9 / (15 – 3)] + 1 = 1.75$
- Officer's $Preference_{11} = 11 – 2 = 9$
- Command's $Preference_{11} = 0$
- $H_{11} = 1 + (c_1 \times Language_{11}) + (c_2 \times Credentials_{11}) + (c_3 \times Experience_{11}) + (c_4 \times$ Officer's $Preference_{11}) + (c_5 \times$ Command's $Preference_{11}) = 1 + (1 \times 1.75) + (1 \times 4) + (1 \times 1.75) + (2 \times 9) + (2 \times 0) = 26.5$.

The computation of the H_{ij} values is done with the ksergis.dec_H_Function stored procedure. Also, the ksergis.dec_H_Fill stored procedure stores these H_{ij} values in the H table described in the previous chapter. Both of these procedures are presented in the Appendix.

The nature of the utility function needs H_{ij} values in the real interval [1,10], so the H_{ij} values need to be 'normalized' between these two limits. In order to perform this 'normalization', the maximum H_{ij} value among all the O_j officers per each J_i job is first stored in the MAX VALUE ALL JOBS table described in the previous chapter. This table contains the max $(H_{.j})$ for every J_i job. Then, for each O_j officer every H_{ij} value is normalized using the following function.

$$H_{ij} = [H_{ij} \times 9 / \max (H_{.j})] + 1$$

The ksergis.dec_H_Normalize stored procedure performs this conversion and the new H_{ij} value is stored back to the H table. This procedure is presented in the Appendix.

Take the last example and assume that $\max (H_{.1}) = 28$. Since the H_{11} value is 26.5, the new H_{11} value is the following:

$$H_{11} = [H_{11} \times 9 / \max (H_{.1})] + 1 = [26.5 \times 9 / 28] + 1 = 9.5178$$

B. ALGORITHM

The philosophy of the algorithm is greedy choice. It tries to pick the maximum H_{ij} value from the remaining O_j officers per J_i job, beginning from the job with the highest priority through the job with the lowest one. At the same time, it tries to minimize the number of unassigned jobs.

The algorithm uses the following tables.

1. H Table

The H table contains the Job (JobId, PlaceCode as described in Chapter 3), the Officer and the corresponding HValue.

A visual representation is shown on the table below. Every H_{ij} value is a number between 1 and 10. There could be cells with NULL values as it was mentioned before.

	J_1	**J_2**	**...**	**J_n**
O_1	H_{11}	H_{12}	...	H_{1n}
O_2	H_{21}	H_{22}	...	H_{2n}
...	...	...	...	...
O_m	H_{m1}	H_{m2}	...	H_{mn}

2. PRIORITY Table

The PRIORITY table contains the Job (JobId, PlaceCode), the Detailer's Priority (the JOB entity's Priority - different per JobId as described in Chapter 3), the overall Priority (a Counter that describes the sorting order of each JobId, PlaceCode pair according to the Detailer's Priority) and a Flag. An example is shown in the table below.

Job	Detailer’s Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	1
J_4	9	4	1
J_5	9	5	0
J_6	**8**	**6**	**0**
J_7	7	7	0
...	...	...	...
J_n	4	n	0

3. MAX VALUE Table

The MAX VALUE table contains the Job (JobId, PlaceCode), the Officer (ApplicantId) and the H_{ij} max value (MAXValue), a value that is selected after the algorithm completes the jobs-to-officers distribution. ApplicantId corresponds to the officer who has the MAXValue for the specific Job-Platform/Base pair. An example is shown on the table below.

Job	Officer	H_{ij} max value
J_1	O_7	6.83
J_2	O_2	8.76
...	...	...
J_n	O_k	9.52

4. USED APPLICANTS Table

The USED APPLICANTS table contains the Job (JobId, PlaceCode) and the Officer (ApplicantId). This entity contains the officers of the used max H_{ij} values per job J_i, while the algorithm checks for any available max value on the J_{i+1} job. An example is shown in the table below.

Job	Officer
J_1	O_1, O_3, O_5
J_2	O_2
J_3	O_4, O_6

5. ASSIGNED APPLICANTS Table

The ASSIGNED APPLICANTS table contains the Officers (ApplicantId) that have been already assigned to jobs. An example is shown on the table below.

Officer
O_7
O_8

6. DELETED JOBS Table

The DELETED JOBS table contains the Jobs (JobId, PlaceCode) for which a match cannot be found. An example is shown on the table below.

Job
J_4
J_6

Before presenting the algorithm, there is a need to present a predicate that will be used extensively in the algorithm. V_i contains all the H_{ij} values of the J_i job that are not NULL and the corresponding O_j officers do not belong to either the ASSIGNED APPLICANTS table nor the USED APPLICANTS table. V_i = {{H_{ij} ? H for J_i job (excluding NULL values)} – {H_{ij} ? H: O_j ? ASSIGNED APPLICANTS table} – { H_{ij} ? H: O_j ? USED APPLICANTS table for J_i job}}

Algorithm:

i refers to Priority of job J_i

Step 1: Compute the PRIORITY table and fill the Flag entries with 0.

Step 2: Compute the H table.

Step 3: Delete the jobs on the PRIORITY table that have only null values on the H table (adjust the Priority numbers on the Priority table) and populate the DELETED JOBS table.

Step 4: i -> 1

Step 5: WHILE (i <= PRIORITY table length)

BEGIN

Step 6: Calculate V_i

Step 7: IF (i =1) AND ($Flag_i$ = 1) AND (V_1 = 0) THEN

BEGIN

Step 8: Delete Higher Priority Job (lowest Priority number) with Flag = 0

Step 9: Recalculate PRIORITY table length

Step 10: Delete all J_1 entries from the USED APPLICANTS table

Step 11: Recalculate V_1

END

Step 12: IF (V_i ? 0) THEN

BEGIN

Step 13: Compute MAX(H_{ik}) from the V_i set

Step 14: Input MAX(H_{ik}), O_k in the MAX VALUE table for job J_i

Step 15: Input O_k in the ASSIGNED APPLICANTS table

Step 16: $Flag_i$ -> 1

Step 17: i -> i + 1

END

Step 18: ELSE

BEGIN

Step 19: Delete $H_{i-1,r}$ and P_r from the MAX VALUE table for job J_{i-1}

Step 20: Delete O_r from the ASSIGNED APPLICANTS table

Step 21: Input O_r in the USED APPLICANTS table for job J_{i-1}

Step 22: Delete all J_i entries from the USED APPLICANTS table

Step 23: i -> i - 1

END

END

The following example considers the case when there are five officers to be assigned to 6 Jobs. For this demonstration and in order to keep it simple, the H_{ij} values are considered to be positive numbers with no upper bound limit.

After **Step 1** the PRIORITY table is:

Job	**Detailer's Priority**	**Priority**	**Flag**
J_1	10	1	0
J_2	10	2	0
J_3	9	3	0
J_4	8	4	0
J_5	8	5	0
J_6	7	6	0

Suppose that after **Step 2** the H table looks like:

	$\mathbf{J_1}$	$\mathbf{J_2}$	$\mathbf{J_3}$	$\mathbf{J_4}$	$\mathbf{J_5}$	$\mathbf{J_6}$
O_1	10					
O_2	20	40	15		60	
O_3		35				
O_4						
O_5						40

The empty cells are NULL values.

After **Step 3** the H table becomes:

	J_1	J_2	J_3	J_5	J_6
O_1	10				
O_2	20	40	15	60	
O_3		35			
O_4					
O_5					40

The Priority table becomes:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	0
J_2	10	2	0
J_3	9	3	0
J_5	8	4	0
J_6	7	5	0

And the DELETED JOBS table becomes:

Jobs	J_4

After **Step 4:** i = 1

After **Step 5:** WHILE (1 <= 5)

After **Step 6:** $V_1 = \{\{10, 20\} - 0 - 0\} = \{10, 20\}$

Step 7 IF statement is False since $Flag_1 = 0$ and V_1 ? 0

Step 12 IF statement is True since V_1 ? 0

After **Step 13:** $MAX(H_{1k}) = H_{12} = 20$ for O_2

After **Step 14** 20, O_2 are put in the MAX VALUE table for job J_1

MAX Value table:

Job	**Officer**	**max value H_{ij}**
J_1	O_2	20

After **Step 15** O2 is put in the Assigned Applicants table

Assigned Applicants table:

Officer	O_2

After **Step 16** $Flag_1 = 1$

The PRIORITY table becomes:

Job	**Detailer's Priority**	**Priority**	**Flag**
J_1	10	1	1
J_2	10	2	0
J_3	9	3	0
J_5	8	4	0
J_6	7	5	0

After **Step 17** i = 2

After **Step 5:** WHILE (2 <= 5)

After **Step 6:** $V_2 = \{\{40, 35\}-\{40\} - 0\} = \{35\}$

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** $MAX(H_{2k}) = H_{23} = 35$ for O_3

After **Step 14** 35, O_3 are put in the MAX VALUE table for job J_2

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_2	20
J_2	O_3	35

After **Step 15** O_3 is put in the ASSIGNED APPLICANTS table

Assigned Applicants table:

Officer	O_2, O_3

After **Step 16** $Flag_2 = 1$

The PRIORITY table becomes:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	0
J_5	8	4	0
J_6	7	5	0

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 5)

After **Step 6:** $V_3 = \{\{15\}-\{15\} - 0\} = 0$

Step 7 IF statement is False since i = 3

Step 12 IF statement is False since $V_3 = 0$

Step 18 Else statement is True

After **Step 19** H_{23} and O_3 are deleted from the MAX VALUE table for job J_2

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_2	20

After **Step 20** O_3 is deleted from the ASSIGNED APPLICANTS table

Assigned Applicants table:

Officer	O_2

After **Step 21** O_3 is put in the USED APPLICANTS table for job J_2

USED APPLICANTS table:

Job	**Officer**
J_2	O_3

After **Step 22** all J_3 entries are deleted from the USED APPLICANTS table. In this case there is no entry for J_3

After **Step 23** i = 2

After **Step 5:** WHILE (2 <= 5)

After **Step 6:** $V_2 = \{\{40, 35\}-\{40\}-\{35\}\} = 0$

Step 7 IF statement is False since i = 2

Step 12 IF statement is False since $V_2 = 0$

Step 18 Else statement is True

After **Step 19** H_{12} and O_2 are deleted from the MAX VALUE table for job J_1. The MAX VALUE table is empty

After **Step 20** O_2 is deleted from the ASSIGNED APPLICANTS table. The ASSIGNED APPLICANTS table is empty

After **Step 21** O_2 is put in the USED APPLICANTS table for job J_1

Used Applicants table:

Job	**Officer**
J_1	O_2
J_2	O_3

After **Step 22** all J_2 entries are deleted from the USED APPLICANTS table.

Used Applicants table:

Job	Officer
J_1	O_2

After **Step 23** i = 1

After **Step 5:** WHILE (1 <= 5)

After **Step 6:** V_1 = {{10, 20}- 0 - {20}} = 10

Step 7 IF statement is False since V_1 ? 0

Step 12 IF statement is True since V_1 ? 0

After **Step 13:** MAX(H_{1k}) = H_{11} = 10 for O_1

After **Step 14** 10, O_1 are put in the MAX VALUE table for job J_1

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_1	10

After **Step 15** O1 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1

After **Step 16** $Flag_1$ = 1

After **Step 17** i = 2

After **Step 5:** WHILE (2 <= 5)

After **Step 6:** V_2 = {{40, 35}- 0 - 0} = {40, 35}

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** MAX(H_{2k}) = H_{22} = 40 for O_2

After **Step 14** 40, O_2 are put in the MAX VALUE table for job J_2

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_2	40

After **Step 15** O_2 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_2

After **Step 16** $Flag_2 = 1$

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 5)

After **Step 6:** $V_3 = \{\{15\}-\{15\} - 0\} = 0$

Step 7 IF statement is False since i = 3

Step 12 IF statement is False since $V_3 = 0$

Step 18 Else statement is True

After **Step 19** H_{22} and O_2 are deleted from the MAX VALUE table for job J_2

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_1	10

After **Step 20** O_2 is deleted from the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1

After **Step 21** O_2 is put in the USED APPLICANTS table for job J_2

USED APPLICANTS table:

Job	Officer
J_1	O_2
J_2	O_2

After **Step 22** all J_3 entries are deleted from the USED APPLICANTS table. In this case there is no entry for J_3

After **Step 23** i = 2

After **Step 5:** WHILE (2 <= 5)

After **Step 6:** $V_2 = \{\{40, 35\} - 0 - \{40\}\} = 35$

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** $MAX(H_{2k}) = H_{23} = 35$ for O_3

After **Step 14** 35, O_3 are put in the MAX VALUE table for job J_2

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_3	35

After **Step 15** O_3 is put in the Assigned Applicants table

Assigned Applicants table:

Officer	O_1, O_3

After **Step 16** $Flag_2 = 1$

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 5)

After **Step 6:** $V_3 = \{\{15\} - 0 - 0\} = 15$

Step 7 IF statement is False since i = 3

Step 12 IF statement is True since V_3 ? 0

After **Step 13:** MAX(H_{3k}) = H_{32} = 15 for O_2

After **Step 14** 15, O_2 are put in the MAX VALUE table for job J_3

MAX Value table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_3	35
J_3	O_2	15

After **Step 15** O_2 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_3, O_2

After **Step 16** $Flag_3$ = 1

The PRIORITY table becomes:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	1
J_5	8	4	0
J_6	7	5	0

After **Step 17** i = 4

After **Step 5:** WHILE (4 <= 5)

After **Step 6:** V_4 = {{60}-{60} - 0} = 0

Step 7 IF statement is False since i = 4

Step 12 IF statement is False since V_4 = 0

Step 18 Else statement is True

After **Step 19** H_{32} and O_2 are deleted from the MAX VALUE table for job J_3.

MAX Value table:

Job	**Officer**	**max value H_{ij}**
J_1	O_1	10
J_2	O_3	35

After **Step 20** O_2 is deleted from the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_3

After **Step 21** O_2 is put in the USED APPLICANTS table for job J_3

USED APPLICANTS table:

Job	**Officer**
J_1	O_2
J_2	O_2
J_3	O_2

After **Step 22** all J_4 entries are deleted from the USED APPLICANTS table. In this case there is no entry for J_4

After **Step 23** i = 3

After **Step 5:** WHILE (3 <= 5)

After **Step 6:** $V_3 = \{\{15\} - 0 - \{15\}\} = 0$

Step 7 IF statement is False since i = 3

Step 12 IF statement is False since $V_3 = 0$

Step 18 Else statement is True

After **Step 19** H_{23} and O_3 are deleted from the MAX VALUE table for job J_2.

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10

After **Step 20** O_3 is deleted from the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1

After **Step 21** O_3 is put in the USED APPLICANTS table for job J_2

USED APPLICANTS table:

Job	Officer
J_1	O_2
J_2	O_2, O_3
J_3	O_2

After **Step 22** all J_3 entries are deleted from the USED APPLICANTS table.

USED APPLICANTS table:

Job	Officer
J_1	O_2
J_2	O_2, O_3

After **Step 23** i = 2

After **Step 5:** WHILE (2 <= 5)

After **Step 6:** $V_2 = \{\{40, 35\} - 0 - \{40, 35\}\} = 0$

Step 7 IF statement is False since i = 2

Step 12 IF statement is False since $V_2 = 0$

Step 18 Else statement is True

After **Step 19** H_{11} and O_1 are deleted from the MAX VALUE table for job J_1. The MAX VALUE table is empty

After **Step 20** O_1 is deleted from the ASSIGNED APPLICANTS table. The ASSIGNED APPLICANTS table is empty

After **Step 21** O_1 is put in the USED APPLICANTS table for job J_1

USED APPLICANTS table:

Job	**Officer**
J_1	O_2, O_1
J_2	O_2, O_3

After **Step 22** all J_2 entries are deleted from the USED APPLICANTS table.

USED APPLICANTS table:

Job	**Officer**
J_1	O_2, O_1

After **Step 23** i = 1

After **Step 5:** WHILE (1 <= 5)

After **Step 6:** $V_1 = \{\{10, 20\} - 0 - \{10, 20\}\} = 0$

Step 7 IF statement is True since i = 1, $Flag_1 = 1$ and $V_1 = 0$

After **Step 8** J_5 is deleted from the PRIORITY table, since it's the Higher Priority Job (lowest Priority number) with Flag = 0

The H table becomes:

	$\mathbf{J_1}$	$\mathbf{J_2}$	$\mathbf{J_3}$	$\mathbf{J_6}$
O_1	10			
O_2	20	40	15	
O_3		35		
O_4				
O_5				40

The PRIORITY table becomes:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	1
J_6	7	4	0

And the DELETED JOBS table becomes:

Jobs	J_4, J_5

After **Step 9** the PRIORITY table length is recalculated to 4

After **Step 10** all J_1 entries are deleted from the USED APPLICANTS table. The USED APPLICANTS table is empty

After **Step 11:** $V_1 = \{\{10, 20\} - 0 - 0\} = \{10, 20\}$

Step 12 IF statement is True since V_1 ? 0

After **Step 13:** $MAX(H_{1k}) = H_{12} = 20$ for O_2

After **Step 14** 20, O_2 are put in the MAX VALUE table for job J_1

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_2	20

After **Step 15** O_2 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_2

After **Step 16** $Flag_1 = 1$

After **Step 17** i = 2

After **Step 5:** WHILE (2 <= 4)

After **Step 6:** $V_2 = \{\{40, 35\}-\{40\} - 0\} = \{35\}$

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** $MAX(H_{2k}) = H_{23} = 35$ for O_3

After **Step 14** 35, O_3 are put in the MAX VALUE table for job J_2

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_2	20
J_2	O_3	35

After **Step 15** O_3 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_2, O_3

After **Step 16** $Flag_2 = 1$

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 4)

After **Step 6:** $V_3 = \{\{15\}-\{15\} - 0\} = 0$

Step 7 IF statement is False since i = 3

Step 12 IF statement is False since $V_3 = 0$

Step 18 Else statement is True

After **Step 19** H_{23} and O_3 are deleted from the MAX VALUE table for job J_2

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_2	20

After **Step 20** O_3 is deleted from the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_2

After **Step 21** O_3 is put in the USED APPLICANTS table for job J_2

USED APPLICANTS table:

Job	**Officer**
J_2	O_3

After **Step 22** all J_3 entries are deleted from the USED APPLICANTS table. In this case there is no entry for J_3

After **Step 23** i = 2

After **Step 5:** WHILE (2 <= 4)

After **Step 6:** $V_2 = \{\{40, 35\}-\{40\}-\{35\}\} = 0$

Step 7 IF statement is False since i = 2

Step 12 IF statement is False since $V_2 = 0$

Step 18 Else statement is True

After **Step 19** H_{12} and O_2 are deleted from the MAX VALUE table for job J_1. The MAX VALUE table is empty

After **Step 20** O_2 is deleted from the ASSIGNED APPLICANTS table. The ASSIGNED APPLICANTS table is empty

After **Step 21** O_2 is put in the USED APPLICANTS table for job J_1

USED APPLICANTS table:

Job	**Officer**
J_1	O_2
J_2	O_3

After **Step 22** all J_2 entries are deleted from the USED APPLICANTS table.

USED APPLICANTS table:

Job	Officer
J_1	O_2

After **Step 23** i = 1

After **Step 5:** WHILE (1 <= 4)

After **Step 6:** $V_1 = \{\{10, 20\}- 0 - \{20\}\} = 10$

Step 7 IF statement is False since V_1 ? 0

Step 12 IF statement is True since V_1 ? 0

After **Step 13:** $MAX(H_{1k}) = H_{11} = 10$ for O_1

After **Step 14** 10, O_1 are put in the MAX VALUE table for job J_1

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10

After **Step 15** O_1 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1

After **Step 16** $Flag_1 = 1$

After **Step 17** i = 2

After **Step 5:** WHILE (2 <= 4)

After **Step 6:** $V_2 = \{\{40, 35\}- 0 - 0\} = \{40, 35\}$

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** $MAX(H_{2k}) = H_{22} = 40$ for O_2

After **Step 14** 40, O_2 are put in the MAX VALUE table for job J_2

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_2	40

After **Step 15** O_2 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_2

After **Step 16** $Flag_2 = 1$

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 4)

After **Step 6:** $V_3 = \{\{15\}-\{15\} - 0\} = 0$

Step 7 IF statement is False since i = 3

Step 12 IF statement is False since $V_3 = 0$

Step 18 Else statement is True

After **Step 19** H_{22} and O_2 are deleted from the MAX VALUE table for job J_2

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10

After **Step 20** O_2 is deleted from the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1

After **Step 21** O_2 is put in the USED APPLICANTS table for job J_2

USED APPLICANTS table:

Job	Officer
J_1	O_2
J_2	O_2

After **Step 22** all J_3 entries are deleted from the USED APPLICANTS table. In this case there is no entry for J_3

After **Step 23** i = 2

After **Step 5:** WHILE (2 <= 4)

After **Step 6:** V_2 = {{40, 35} - 0 -{40}} = 35

Step 7 IF statement is False since i = 2

Step 12 IF statement is True since V_2 ? 0

After **Step 13:** MAX(H_{2k}) = H_{23} = 35 for O_3

After **Step 14** 35, O_3 are put in the MAX VALUE table for job J_2

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_3	35

After **Step 15** O_3 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_3

After **Step 16** $Flag_2$ = 1

After **Step 17** i = 3

After **Step 5:** WHILE (3 <= 4)

After **Step 6:** V_3 = {{15} - 0 - 0} = 15

Step 7 IF statement is False since i = 3

Step 12 IF statement is True since V_3 ? 0

After **Step 13:** MAX(H_{3k}) = H_{32} = 15 for O_2

After **Step 14** 15, O_2 are put in the MAX VALUE table for job J_3

MAX VALUE table:

Job	**Officer**	**max value H_{ij}**
J_1	O_1	10
J_2	O_3	35
J_3	O_2	15

After **Step 15** O_2 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_3, O_2

After **Step 16** $Flag_3$ = 1

After **Step 17** i = 4

After **Step 5:** WHILE (4 <= 4)

After **Step 6:** V_4 = {{40} - 0 - 0} = 40

Step 7 IF statement is False since i = 4

Step 12 IF statement is True since V_4 ? 0

After **Step 13:** MAX(H_{4k}) = H_{45} = 40 for O_5

After **Step 14** 40, O_5 are put in the MAX VALUE table for job J_6

MAX VALUE table:

Job	**Officer**	**max value H_{ij}**
J_1	O_1	10
J_2	O_3	35
J_3	O_2	15
J_6	O_5	40

After **Step 15** O_5 is put in the ASSIGNED APPLICANTS table

ASSIGNED APPLICANTS table:

Officer	O_1, O_3, O_2, O_5

After **Step 16** $Flag_4 = 1$

The PRIORITY table becomes:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	1
J_6	7	4	1

After **Step 17** i = 5

After **Step 5:** WHILE (5 <= 4) is False

…And this is the end of the algorithm.

The results are:

H table:

	J_1	J_2	J_3	J_6
O_1	10			
O_2	20	40	15	
O_3		35		
O_4				
O_5				40

PRIORITY table:

Job	Detailer's Priority	Priority	Flag
J_1	10	1	1
J_2	10	2	1
J_3	9	3	1
J_6	7	4	1

USED APPLICANTS table:

Job	Officer
J_1	O_2
J_2	O_2

ASSIGNED APPLICANTS table:

Officer	O_1, O_3, O_2, O_5

DELETED JOBS table:

Jobs	J_4, J_5

MAX VALUE table:

Job	Officer	max value H_{ij}
J_1	O_1	10
J_2	O_3	35
J_3	O_2	15
J_6	O_5	40

The algorithm and all the supportive code are presented in the Appendices. The code is written in Transact-SQL and is stored in stored procedures.

Before continuing to the description of the utility function, there is a problem that could occur and should be addressed. Consider the following case of 3 jobs to be distributed to 3 officers:

	J_1	**J_2**	**J_3**
O_1	10	10	10
O_2	10	10	9
O_3	10	9	9

The problem is that all the officers have the same maximum HValue (10 for this instance) for J_1 job and 2 of them have the same maximum HValue (10 again) for J_2 job. If the algorithm chooses O_1 officer for J_1 job, then the final distribution will be the following.

Job	**Officer**	**max value H_{ij}**
J_1	O_1	10
J_2	O_2	10
J_3	O_3	9

Apparently, the algorithm made a wrong decision when it picked up O_1 officer for J_1 job. It should pick up O_3 officer for J_1 job first, then O_2 officer for J_2 job and finally O_1 officer for J_3 job. Any other combination does not give the desired outcome. The final distribution will be the following.

Job	**Officer**	**max value H_{ij}**
J_1	O_3	10
J_2	O_2	10
J_3	O_1	10

So, there should be a way to address that problem. The following example will help towards that direction.

	J_1	J_2	J_3	J_4
O_1	10	10	7	10
O_2	10	10	9	8
O_3	10	9	8	7
O_4	10	8	10	10

The following tables are constructed in order to help the algorithm to make the correct decision.

a. Same Max Value

This table stores the Officers that have the same max HValue. For the above example it will store the O_1, O_2, O_3 and O_4 officers since they all have the same max HValue 10.

b. Min Value Applicants

Looking carefully at the above example, the O_3 officer row for jobs J_2, J_3 and J_4, does not contain any max HValue like the rest rows have. For example O_2 officer row has 1 max HValue (10) under job J_2, O_1 officer row has 2 max HValues (10) under jobs J_2 and J_4, and finally O_4 officer row has 2 max HValues (10) under jobs J_3 and J_4. This table stores the O_3 officer and the HValue 9, which is the HValue of the same officer O_3 for the job with the next lower priority (job J_2).

c. Multiple Max Values

Again, looking at the above example job J_2 has 2 max HValues (10) under it, for officers O_1 and O_2. Also, job J_4 has 2 max HValues (10) under it, for officers O_1 and O_4. This table stores these jobs that have multiple max HValues under them, with their corresponding officers. For this instance it stores the J_2, O_1 pair, the J_2, O_2 pair, the J_4, O_1 pair and the J_4, O_4 pair.

d. One Max Value

Again, looking at the above example job J_3 has 1 max HValue (10) under it, for the officer O_4. This table stores these jobs that have only one max HValue under them, with their corresponding officers. For this instance it stores only the J_3, O_4 pair.

The algorithm below, a sub-algorithm of the main one, is solving this problem taking into account the tables just described.

Assume that there are multiple max HValues on the J_i job. The algorithm returns one of these officers (with the same max HValue) for the J_i job.

Algorithm:

Step 1: Fill SAME MAX VALUE table

Step 2: For All officers ? SAME MAX VALUE

BEGIN

Step 3: Find O_j that has **no** max HValue for all jobs J_k with priorities $P_k < P_i$

Step 4: For this O_j select $H_{j,\ i+1}$, where J_{i+1} is the job with the next lower priority of job J_i

Step 5: Input $H_{j,\ i+1}$ and O_j in MIN VALUE APPLICANTS table

END

Step 6: IF (MIN VALUE APPLICANTS ? 0)

BEGIN

Step 7: Select O_m, with min ($H_{m,\ i+1}$) where O_m and $H_{m,i+1}$? MIN VALUE APPLICANTS

Step 8: Return O_m

END

Step 9: ELSE

BEGIN

Step 10: For All jobs J_k with priorities $P_k < P_i$

BEGIN

Step 11: Find J_p jobs that have **one** max HValue for all the rest officers

Step 12: Find correspondent officer O_s

Step 13: Find J_q jobs that have **multiple** max HValues for all the rest officers

Step 14: Find correspondent officers O_t

Step 15: Input J_p, O_s pair in ONE MAX VALUE table

Step 16: Input J_q, O_t pairs in MULTIPLE MAX VALUES table

END

Step 17: IF (MULTIPLE MAX VALUES ? 0)

BEGIN

Step 18: IF **rest** jobs J_k with priorities $P_k < P_i$ are more than 2

BEGIN

Step 19: Select O_s with min ($H_{s, i+1}$), where O_s ? MULTIPLE MAX VALUES

Step 20: IF (min ($H_{s, i+1}$) < max ($H_{j, i+1}$) for all O_j)

BEGIN

Step 21: Return O_s

END

Step 22: ELSE

BEGIN

Step 23: Select officer O_s ? MULTIPLE MAX VALUES with $H_{s, i+1}$ = max ($H_{j, i+1}$) that has the least number of max HValues beyond J_{i+1} AND the job that has one of these max HValues, has the **lowest** priority.

Step 24: Return O_s

END

END

Step 25: ELSE IF **rest** jobs J_k with priorities $P_k < P_i$ are 2

BEGIN

Step 26: Select O_s with min ($H_{s, i+2}$), where O_s ? MULTIPLE MAX VALUES

Step 27: Return O_s

END

Step 28: ELSE IF **rest** jobs J_k with priorities $P_k < P_i$ are 1

BEGIN

Step 29: Select O_s with min ($H_{s, i+1}$), where O_s ? MULTIPLE MAX VALUES

Step 30: Return O_s

END

END

Step 31: ELSE IF (MULTIPLE MAX VALUES = 0) AND (ONE MAX VALUE ? 0)

BEGIN

Select O_s ? ONE MAX VALUE, where the correspondent J_k has priority $P_k < P_{i-1}$ and P_k is minimum

Step 32: Return O_s

END

Step 33: ELSE

BEGIN

Step 34: Choose O_s randomly

Step 35: Return O_s

END

END

The following examples demonstrate the use of the algorithm.

Example 1:

	J_1	**J_2**	**J_3**	**J_4**
O_1	10	10	7	9
O_2	10	8	9	10
O_3	10	7	8	10
O_4	10	6	10	8

After **Step 1** the SAME MAX VALUE table becomes

SAME MAX VALUE table:

Officer	O_1, O_2, O_3, O_4

Step 6 statement is False since beyond J_1 job, O_1 officer row has 1 max HValue (10) under job J_2, O_2 officer row has 1 max HValue (10) under job J_4, O_3 officer row has 1 max HValue (10) under job J_4 too, and finally O_4 officer row has 1 max HValue (10) under job J_3.

After the loop from **Step 10** to **Step 16**, we have:

MULTIPLE MAX VALUES table:

Job	**Officer**
J_4	O_2
J_4	O_3

ONE MAX VALUE table:

Job	**Officer**
J_2	O_1
J_3	O_4

Step 17 is true (MULTIPLE MAX VALUES ? 0)

Step 18 is true since the jobs J_k with priorities $P_k < P_1$ are more than 2 (these are J_2, J_3, J_4).

After **Step 19** the min ($H_{s,\ i+1}$) is $H_{32} = 7$ of O_3, since for officers O_2, O_3 ? MULTIPLE MAX VALUES, $H_{32} = 7 < 8 = H_{22}$.

After **Step 21** the algorithm is ended and officer O_3 is returned.

Example 2:

	$\mathbf{J_1}$	$\mathbf{J_2}$	$\mathbf{J_3}$	$\mathbf{J_4}$
$\mathbf{O_1}$	10	10	7	10
$\mathbf{O_2}$	10	8	9	7
$\mathbf{O_3}$	10	7	8	8
$\mathbf{O_4}$	10	6	10	10

After **Step 1** the SAME MAX VALUE table becomes

SAME MAX VALUE table:

Officer	O_1, O_2, O_3, O_4

After the loop from **Step 2** to **Step 5**, we have:

MIN VALUE APPLICANTS table:

Officer	**HValue**
O_2	8
O_3	7

Step 6 statement is True

After **Step 7** officer O_3 is selected since $H_{32} = 7 < 8 = H_{22}$

After **Step 8** the algorithm is ended and officer O_3 is returned.

Example 3:

	$\mathbf{J_1}$	$\mathbf{J_2}$	$\mathbf{J_3}$	$\mathbf{J_4}$
$\mathbf{O_1}$	10	10	7	7
$\mathbf{O_2}$	10	8	10	8
$\mathbf{O_3}$	10	7	9	10
$\mathbf{O_4}$	10	10	8	9

After **Step 1** the SAME MAX VALUE table becomes

SAME MAX VALUE table:

Officer	O_1, O_2, O_3, O_4

Step 6 statement is False since beyond J_1 job, O_1 officer row has 1 max HValue (10) under job J_2, O_2 officer row has 1 max HValue (10) under job J_3, O_3 officer row has 1 max HValue (10) under job J_4, and finally O_4 officer row has 1 max HValue (10) under job J_2.

After the loop from **Step 10** to **Step 16**, we have:

MULTIPLE MAX VALUES table:

Job	**Officer**
J_2	O_1
J_2	O_4

ONE MAX VALUE table:

Job	**Officer**
J_3	O_2
J_4	O_3

Step 17 is true (MULTIPLE MAX VALUES ? 0)

Step 18 is true since the jobs J_k with priorities $P_k < P_1$ are more than 2 (these are J_2, J_3, J_4).

After **Step 19** the min ($H_{s,\ i+1}$) = 10, since for officers O_1, O_4 ? MULTIPLE MAX VALUES, $H_{12} = H_{42} = 10$.

Step 20 is false since min ($H_{s,\ i+1}$) = 10 = max ($H_{j,\ i+1}$)

Step 22 is true

After **Step 23** officers O_1 and O_4 have no max HValue beyond job J_2 for each individual row.

After **Step 24** the algorithm is ended and officer O_4 is returned.

In the next section the Utility Function is described in full detail.

C. UTILITY FUNCTION

The Utility Function tries to capture the concept and philosophy of the algorithm and express it in a mathematical model. The Utility Function helps the detailer to evaluate any changes he/she makes on the solution set and compare the change with the result of the algorithm.

The Utility Function should be a summation of factors that will express both the priority of the J_i job and the H_{ij} value that is selected for that job.

$$\text{Utility Function} = \sum_{i=1}^{n} \text{Factor}_{ij} \quad \textbf{(1)}$$

and

$\text{Factor}_{ij} = \text{Function}(P_i, H_{ij})$ **(2)**

Factor_{ij} is a function of the priority P_i of the J_i job, H_{ij} is the value of the selected pair of J_i job and O_j Officer, and n is the total number of the selected jobs that form the solution. Intuitively this Factor_{ij} should be the multiplication of the H_{ij} value with the P_i priority. The priority P_i is like a coefficient (weight) that multiplied with the H_{ij} value gives the degree of importance the H_{ij} value is for the entire solution.

$\text{Factor}_{ij} = P_i \text{ x } H_{ij}$ **(3)**

The main idea is that the summation of the factors of two adjacent jobs of the algorithm's solution should always be greater than the summation of the factors of the same adjacent jobs of the changed solution. 'Adjacent jobs' are jobs that their priority has 1 value difference.

In order to explain that better, consider the case of a 2 x 2 matrix of the H table.

	J_2	**J_1**
O_1	H_{21}	H_{11}
O_2	H_{22}	H_{12}

Job J_2 has a priority P_2, which is greater than the priority P_1 of job J_1.

$P_2 > P_1 \Rightarrow P_1 = P_2 - 1$ **(4)**

Suppose that all the H_{ij} values are not NULL and that H_{21} value is greater than H_{22} value and H_{11} value is greater than H_{12} value. The algorithm will pick the H_{21} value first because it belongs to the job with higher priority P_2, and then it will choose the remaining H_{12} value. Below, the H_{ij} values in bold are those that are selected by the algorithm.

	J_2	**J_1**
O_1	**H_{21}**	*H_{11}*
O_2	*H_{22}*	**H_{12}**

$H_{21} > H_{22}$, $H_{11} > H_{12}$ **(5)**

There is only one change that the detailer could make, and that is select the H_{22} value first and then select the remaining H_{11} value (the values in italics in the table above). The Utility Function should give a bigger result value for the algorithm solution, than for the change the detailer makes. The Utility Function result for the two cases is shown below.

Algorithm Solution:

Utility Function = $Factor_{21}$ + $Factor_{12}$ = Function (P_2, H_{21}) + Function (P_1, H_{12}) **(6)**

Detailer Change:

Utility Function = $Factor_{22}$ + $Factor_{11}$ = Function (P_2, H_{22}) + Function (P_1, H_{11}) **(7)**

It should be that:

Utility Function Algorithm Solution > Utility Function Detailer Change => **(8)**

Function (P_2, H_{21}) + Function (P_1, H_{12}) > Function (P_2, H_{22}) + Function (P_1, H_{11}) **(9)**

Apparently, this is very hard to succeed since the value of each factor is relative to the P_i and H_{ij} values. There should be a way to benefit the factor with the higher priority. The factor of the higher priority should be bigger by t times the factor of the next lower priority in order for type (8) to be true.

For the case above, the Utility Function should be the following.

Utility Function = t x $Factor_{2j}$ + $Factor_{1j}$ **(10)**

Type (9) is changed into the following form.

t x Function (P_2, H_{21}) + Function (P_1, H_{12}) > t x Function (P_2, H_{22}) + Function (P_1, H_{11}) **(9a)**

Type (10) gives the Utility Function for 2 jobs. The same concept is generalized for type (1) that gives the Utility Function for n jobs. This is described below.

For the first 2 jobs:

t x Function (P_2, H_{2j}) + Function (P_1, H_{1j}) > t x Function (P_2, H_{2j}) + Function (P_1, H_{1j})

For the subsequent 2 jobs:

t^2 x Function (P_3, H_{3j}) + t x Function (P_2, H_{2j}) > t^2 x Function (P_3, H_{3j}) + t x Function (P_2, H_{2j})

For the subsequent 2 jobs:

t^3 x Function (P_4, H_{4j}) + t^2 x Function (P_3, H_{3j}) > t^3 x Function (P_4, H_{4j}) + t^2 x Function (P_3, H_{3j})

The same procedure is done until the last 2 jobs:

t^{n-1} x Function (P_n, H_{nj}) + t^{n-2} x Function (P_{n-1}, $H_{(n-1)j}$) > t^{n-1} x Function (P_n, H_{nj}) + t^{n-2} x Function (P_{n-1}, $H_{(n-1)j}$)

Type (1a) gives the new form of the Utility Function.

$$\text{Utility Function} = \sum_{i=1}^{n} t^{i-1} \text{x Factor}_{ij} \qquad \textbf{(1a)}$$

=> Utility Function = t^{n-1} x $Factor_{nj}$ + t^{n-2} x $Factor_{(n-1)j}$ +...+ t x $Factor_{2j}$ + $Factor_{1j}$ **(1b)**

Taking type (3) into consideration we have that:

Utility Function = t^{n-1} x P_n x H_{ni} + t^{n-2} x P_{n-1} x $H_{(n-1)i}$ +...+ t x P_2 x H_{2i} + P_1 x H_{1i} **(1c)**

Let's go back to the case of the 2 jobs described above.

	J_2	J_1
O_1	H_{21}	H_{11}
O_2	H_{22}	H_{12}

Combining type (8) with type (1c) we have the following:

Utility Function Algorithm Solution > Utility Function Detailer Change =>

t x P_2 x H_{21} + P_1 x H_{12} > t x P_2 x H_{22} + P_1 x H_{11} **(11)**

The worst case scenario should be one of the following possibilities:

- H_{21} value is the maximum value for the J_2 job, H_{11} value is the maximum value for the J_1 job, H_{22} value is the next maximum value for the J_2 job and H_{12} value is the minimum value for the J_1 job.
- H_{22} value is the minimum value for the J_2 job, H_{11} value is the maximum value for the J_1 job, H_{21} value is the next minimum value for the J_2 job and H_{12} value is the minimum value for the J_1 job.

Now it may be seen why it is important to have maximum and minimum values for the H_{ij} variable. Since the maximum and minimum value for the H_{ij} values is 10 and 1 respectively, the H tables for both possibilities are like the following.

For the first possibility we have:

	J_2	J_1
O_1	10	10
O_2	H_{22}	1

Combining type (11) with type (4) we have the following.

$t \times P_2 \times H_{21} + P_1 \times H_{12} > t \times P_2 \times H_{22} + P_1 \times H_{11} \Rightarrow$

$t \times P_2 \times 10 + P_1 \times 1 > t \times P_2 \times H_{22} + P_1 \times 10 \Rightarrow$ **(11b)**

$t > P_1 \times 9 / P_2 \times (10 - H_{22}) \Rightarrow$

$t > [P_1 / P_2] \times [9 / (10 - H_{22})] \Rightarrow$

$t > [(P_2 - 1) / P_2] \times [9 / (10 - H_{22})]$

Since $(P_2 - 1) / P_2 = 1 - 1/ P_2$, it is sufficient for t to be:

$t = 9 / (10 - H_{22})$ **(12)**

For the second possibility we have:

	J_2	J_1
O_1	H_{21}	10
O_2	1	1

Combining type (11) with type (4) we have the following.

$t \times P_2 \times H_{21} + P_1 \times H_{12} > t \times P_2 \times H_{22} + P_1 \times H_{11} \Rightarrow$

$t \times P_2 \times H_{21} + P_1 \times 1 > t \times P_2 \times 1 + P_1 \times 10 \Rightarrow$

$t > P_1 \times 9 / P_2 \times (H_{21} - 1) \Rightarrow$

$t > [P_1 / P_2] \times [9 / (H_{21} - 1)] \Rightarrow$

$t > [(P_2 - 1) / P_2] \times [9 / (H_{21} - 1)]$

Since $(P_2 - 1) / P_2 = 1 - 1/ P_2$, it is sufficient for t to be:

$t = 9 / (H_{21} - 1)$ **(12a)**

So, in both possibilities t is a function of the maximum value and the next most maximum value, or a function of the minimum value and the next most minimum value.

In order to have a unique t value, the maximum and the next most maximum value of all the H_{ij} variables are computed, and are used for this project. In the extreme case that the maximum value and the next most maximum value are the same, then there are several best solutions.

$t = 9 / (\max(H_{ij}) - \text{next max}(H_{ij}))$ **(12b)**

It is obvious that as next max (H_{ij}) approaches the max (H_{ij}), the t value increases infinitely. Things become worse, since t is to the power of (i – 1) and then multiplied by P_i and H_{ij} as type (1c) shows. This means that the result of the Utility Function would be too big for a computer to handle. One solution would be to compute the logarithm of the factor $t^{i-1} \times P_i \times H_{ij}$. But the logarithm of each factor does not provide any solution. Take type (12b), but with the use of logarithms instead.

$\log_{10}(t \times P_2 \times 10) + \log_{10}(P_1 \times 1) > \log_{10}(t \times P_2 \times H_{22}) + \log_{10}(P_1 \times 10) \Rightarrow$

$\log_{10}(t) + \log_{10}(P_2) + \log_{10}(10) + \log_{10}(P_1) > \log_{10}(t) + \log_{10}(P_2) + \log_{10}(H_{22}) + \log_{10}(P_1) + \log_{10}(10) \Rightarrow$

$\log_{10}(H_{22}) < 0$

The last is impossible since:

$H_{22} > 1 \Rightarrow \log_{10}(H_{22}) > \log_{10}(1) = 0.$

In order to avoid this problem, the logarithm of the summation of every 2 subsequent factors is used.

For the first 2 jobs we have that $t \times P_2 \times H_{2j} + P_1 \times H_{1j} > t \times P_2 \times H_{2j} + P_1 \times H_{1j}$. Since both summations are numbers greater or equal to 1, logarithms can be put around them. So we have that $\log_{10}(t \times P_2 \times H_{2j} + P_1 \times H_{1j}) > \log_{10}(t \times P_2 \times H_{2j} + P_1 \times H_{1j})$, which is true.

It is true for the subsequent 2 jobs:

$t^2 \times P_3 \times H_{3j} + t \times P_2 \times H_{2j} > t^2 \times P_3 \times H_{3j} + t \times P_2 \times H_{2j} \Rightarrow$

$\log_{10}(t^2 \times P_3 \times H_{3j} + t \times P_2 \times H_{2j}) > \log_{10}(t^2 \times P_3 \times H_{3j} + t \times P_2 \times H_{2j})$

It is true for next the subsequent 2 jobs:

$t^3 \times P_4 \times H_{4j} + t^2 \times P_3 \times H_{3j} > t^3 \times P_4 \times H_{4j} + t^2 \times P_3 \times H_{3j} \Rightarrow$

$\log_{10}(t^3 \times P_4 \times H_{4j} + t^2 \times P_3 \times H_{3j}) > \log_{10}(t^3 \times P_4 \times H_{4j} + t^2 \times P_3 \times H_{3j})$

It is true for the last 2 jobs too:

$t^{n-1} \text{ x } P_n \text{ x } H_{nj} + t^{n-2} \text{ x } P_{n-1} \text{ x } H_{(n-1)j} > t^{n-1} \text{ x } P_n \text{ x } H_{nj} + t^{n-2} \text{ x } P_{n-1} \text{ x } H_{(n-1)j} \Rightarrow$

$\log_{10}(t^{n-1} \text{ x } P_n \text{ x } H_{nj} + t^{n-2} \text{ x } P_{n-1} \text{ x } H_{(n-1)j}) > \log_{10}(t^{n-1} \text{ x } P_n \text{ x } H_{nj} + t^{n-2} \text{ x } P_{n-1} \text{ x } H_{(n-1)j})$

All these result to the final form of the Utility Function, which is:

$$\text{Utility Function} = \sum_{i=2}^{n} \log_{10}(t^{i-1} \text{ x } P_i \text{ x } H_{ij} + t^{i-2} \text{ x } P_{i-1} \text{ x } H_{(i-1)j)},$$

where t = 9 / (max (H_{ij}) – next max (H_{ij})).

The priorities P_i are stored in the COUNTER table, while the H_{ij} values are stored in the H table. The result of the Utility Function is stored in the ESTIMATE FUNCTION RESULT table. The changes the detailer makes from the MAX VALUE table (the table that stores the algorithm's solution), are stored in the MANIPULATE SOLUTION. Any job and officer the detailer changes from the MANIPULATE SOLUTION table, is stored in the DELETED JOBS MANIPULATE and UNASSIGNED APPLICANTS MANIPULATE table respectively.

Actually, the ESTIMATE FUNCTION RESULT table stores the difference of the Utility Function results from the MAX VALUE and MANIPULATE SOLUTION table. So, if for example the result of the Utility Function for the algorithm's solution is 40 and the result of the Utility Function for the detailer's change is 30, the value that is stored in the ESTIMATE FUNCTION RESULT table is 10.

When the detailer is ready to make a decision, the MAX VALUE table's data or the MANIPULATE SOLUTION table's data are stored in the ASSIGNMENT table.

The Transact-SQL code of the Utility Function and all the supportive sub-procedures are presented in the Appendices.

D. TEST RESULTS

In order to test the algorithm and the Utility Function, tests have been planned and executed. These tests are based on the following issues.

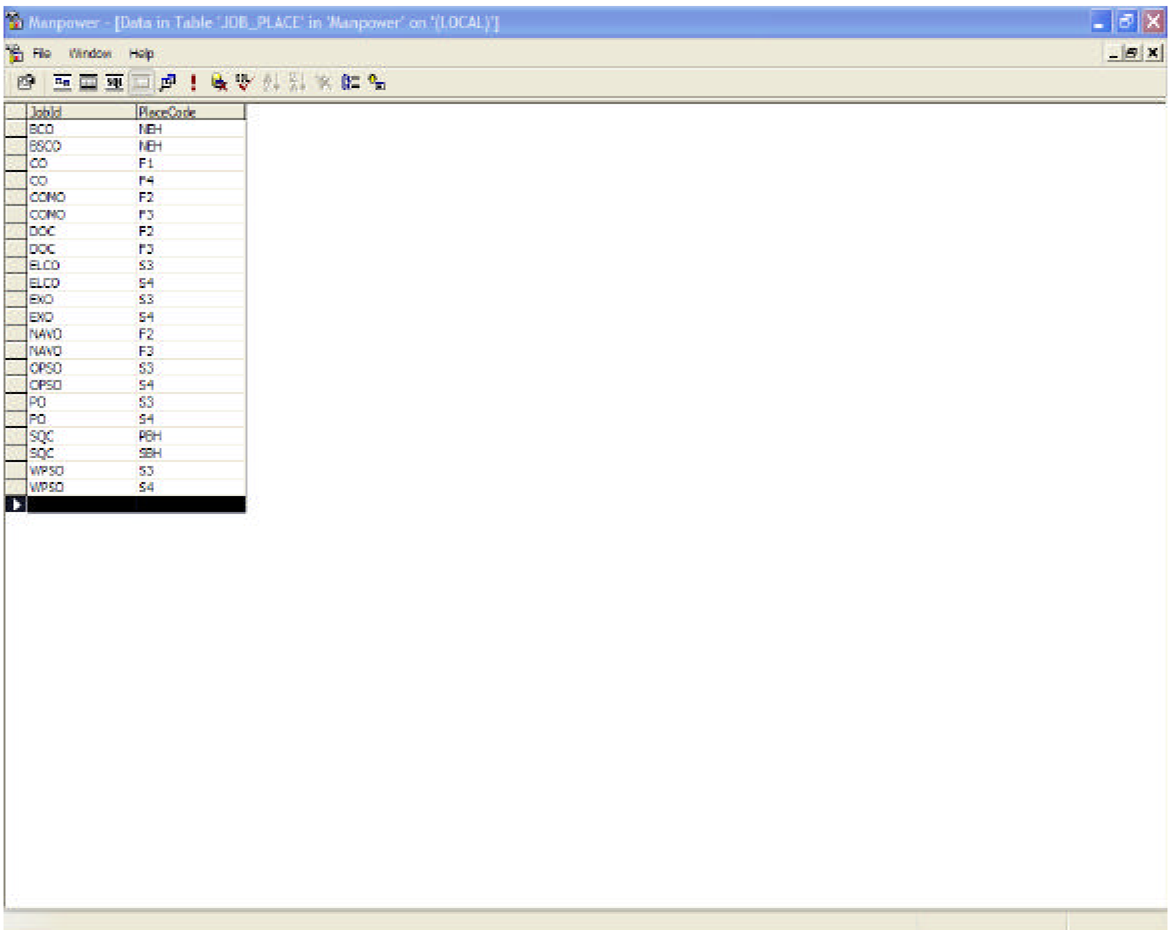
- Estimation of the time length that the computer spends running the algorithm in order to find a distribution.

- Increases on the result that is stored in the ESTIMATE FUNCTION RESULT table, when changes are made on the algorithm's solution.
- Changes on the distribution of the algorithm, when different coefficient weights for the decision variables are given.

A description of the tests is provided below, based on the issues above.

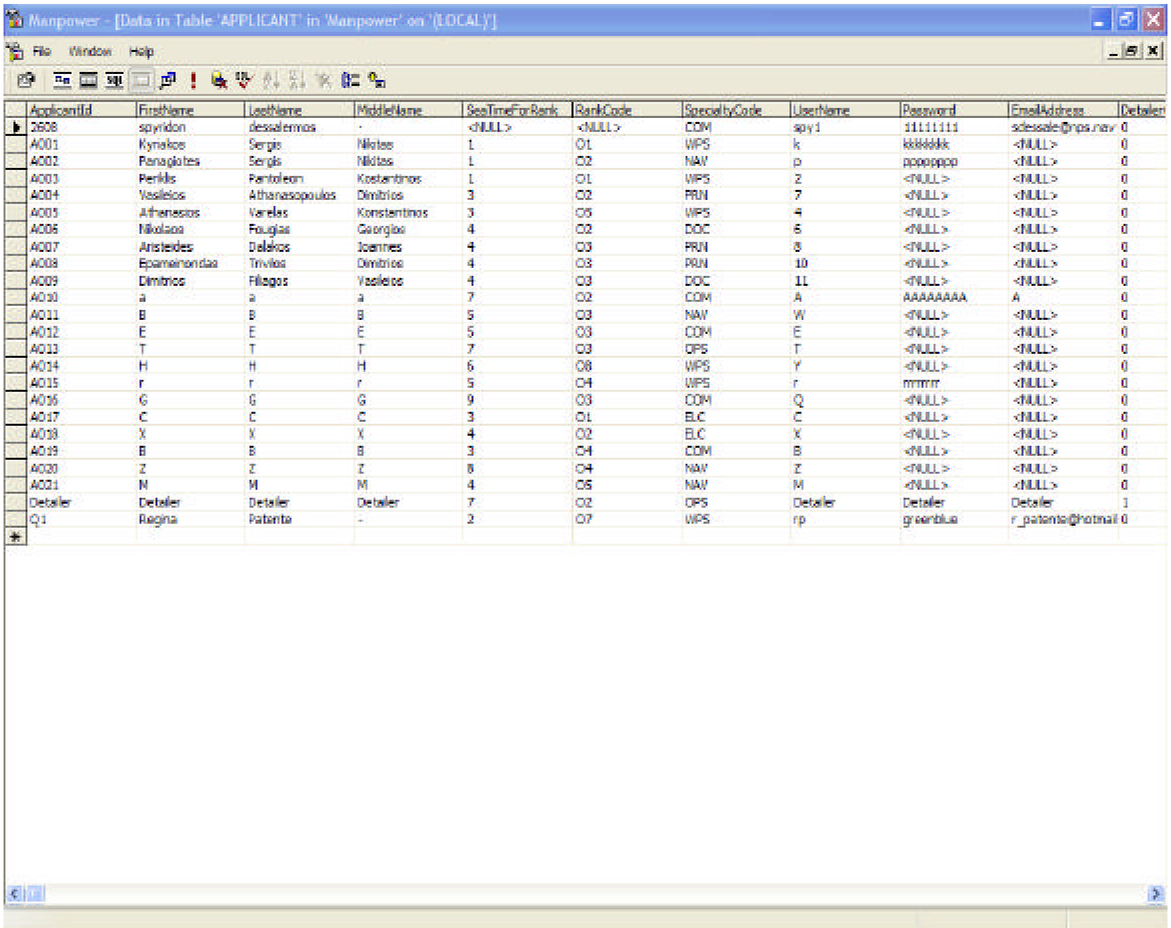
1. Time Length Estimation

The following test considers 22 jobs and 24 officers. The algorithm takes 9 seconds to run and give a distribution. Below are the results.



JobId	PlaceCode
BCO	NEH
BSCO	NEH
CO	F1
CO	F4
COMO	F2
COMO	F3
DOC	F2
DOC	F3
ELCO	S3
ELCO	S4
EXO	S3
EXO	S4
NAVO	F2
NAVO	F3
OPSO	S3
OPSO	S4
PO	S3
PO	S4
SQC	PEH
SQC	SEH
WPSO	S3
WPSO	S4

Figure 25. Job-Platform Pairs to be Fulfilled-Manpower Database.



ApplicantId	FirstName	LastName	MiddleName	SeaTimeForRank	RankCode	SpecialtyCode	UserName	Password	EmailAddress	Detailer
2608	spyridon	dessalemos	-	<NULL>	<NULL>	COM	spy1	11111111	sdessale@nps.nav	0
A001	Kyriakos	Sergis	Nikitas	1	O1	WPS	k	kkkkkkkk	<NULL>	0
A002	Panagiotes	Sergis	Nikitas	1	O2	NAV	p	pppppppp	<NULL>	0
A003	Periklis	Pantoleon	Kostantinos	1	O1	WPS	2	<NULL>	<NULL>	0
A004	Vasileios	Athanasopoulos	Dimitrios	3	O2	PRN	7	<NULL>	<NULL>	0
A005	Athanasios	Varelas	Konstantinos	3	O5	WPS	4	<NULL>	<NULL>	0
A006	Nikolaos	Fougias	Georgios	4	O2	DOC	6	<NULL>	<NULL>	0
A007	Aristeides	Delakos	Ioannes	4	O3	PRN	8	<NULL>	<NULL>	0
A008	Epameinondas	Trivilos	Dimitrios	4	O3	PRN	10	<NULL>	<NULL>	0
A009	Dimitrios	Filagos	Vasileios	4	O3	DOC	11	<NULL>	<NULL>	0
A010	a	a	a	7	O2	COM	A	AAAAAAAA	A	0
A011	B	B	B	5	O3	NAV	W	<NULL>	<NULL>	0
A012	E	E	E	5	O3	COM	E	<NULL>	<NULL>	0
A013	T	T	T	7	O3	OPS	T	<NULL>	<NULL>	0
A014	H	H	H	6	O8	WPS	Y	<NULL>	<NULL>	0
A015	r	r	r	5	O4	WPS	r	rrrrrrr	<NULL>	0
A016	G	G	G	9	O3	COM	Q	<NULL>	<NULL>	0
A017	C	C	C	3	O1	ELC	C	<NULL>	<NULL>	0
A018	X	X	X	4	O2	ELC	X	<NULL>	<NULL>	0
A019	B	B	B	3	O4	COM	B	<NULL>	<NULL>	0
A020	Z	Z	Z	8	O4	NAV	Z	<NULL>	<NULL>	0
A021	M	M	M	4	O6	NAV	M	<NULL>	<NULL>	0
Detailer	Detailer	Detailer	Detailer	7	O2	OPS	Detailer	Detailer	Detailer	1
Q1	Regina	Patente	-	2	O7	WPS	rp	greenblue	r_patente@hotmail	0

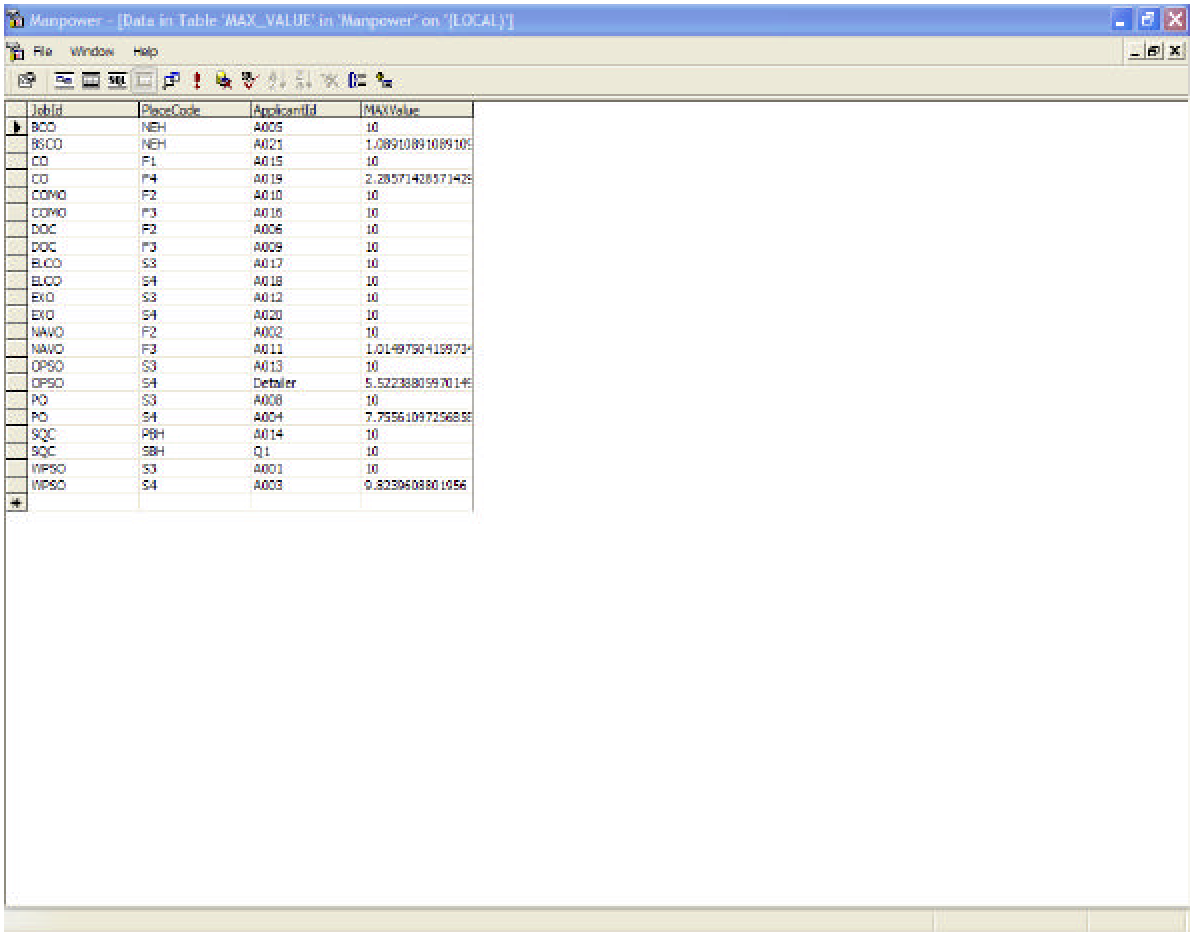
Figure 26. Officers To Be Assigned to the Job-Platform Pairs Above-Manpower Database.

Manpower - [Data in Table 'H' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	ApplicantId	PlaceCode	HValue
BCO	2608	NEH	<NULL>
BCO	A001	NEH	<NULL>
BCO	A002	NEH	<NULL>
BCO	A003	NEH	<NULL>
BCO	A004	NEH	<NULL>
BCO	A005	NEH	10
BCO	A006	NEH	<NULL>
BCO	A007	NEH	<NULL>
BCO	A008	NEH	<NULL>
BCO	A009	NEH	<NULL>
BCO	A010	NEH	<NULL>
BCO	A011	NEH	<NULL>
BCO	A012	NEH	<NULL>
BCO	A013	NEH	<NULL>
BCO	A014	NEH	<NULL>
BCO	A015	NEH	<NULL>
BCO	A016	NEH	<NULL>
BCO	A017	NEH	<NULL>
BCO	A018	NEH	<NULL>
BCO	A019	NEH	<NULL>
BCO	A020	NEH	<NULL>
BCO	A021	NEH	10
BCO	Detailer	NEH	<NULL>
BCO	Q1	NEH	<NULL>
BSCO	2608	NEH	<NULL>
BSCO	A001	NEH	<NULL>
BSCO	A002	NEH	<NULL>
BSCO	A003	NEH	<NULL>
BSCO	A004	NEH	<NULL>
BSCO	A005	NEH	10
BSCO	A006	NEH	<NULL>
BSCO	A007	NEH	<NULL>
BSCO	A008	NEH	<NULL>
BSCO	A009	NEH	<NULL>
BSCO	A010	NEH	<NULL>
BSCO	A011	NEH	<NULL>
BSCO	A012	NEH	<NULL>
BSCO	A013	NEH	<NULL>
BSCO	A014	NEH	<NULL>
BSCO	A015	NEH	1.0891089108910
BSCO	A016	NEH	<NULL>
BSCO	A017	NEH	<NULL>
BSCO	A018	NEH	<NULL>
BSCO	A019	NEH	1.0891089108910
BSCO	A020	NEH	1.0891089108910
BSCO	A021	NEH	1.0891089108910

Figure 27. H Table (Only the First 44 Out of 528 Records Are Shown)-Manpower Database.



Manpower - [Data in Table 'MAX_VALUE' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	PlaceCode	ApplicantId	MAXValue
BCO	NEH	A005	10
BSCO	NEH	A021	1.0891089108910
CO	F1	A015	10
CO	F4	A019	2.285714285714
COMO	F2	A010	10
COMO	F3	A016	10
DOC	F2	A006	10
DOC	F3	A009	10
ELCO	S3	A017	10
ELCO	S4	A018	10
EXO	S3	A012	10
EXO	S4	A020	10
NAVO	F2	A002	10
NAVO	F3	A011	1.01497504159
OPSO	S3	A013	10
OPSO	S4	Detailer	5.5223880597014
PO	S3	A008	10
PO	S4	A004	7.7556109725683
SQC	PBH	A014	10
SQC	SBH	Q1	10
WPSO	S3	A001	10
WPSO	S4	A003	0.823960880 1956

Figure 28. The Solution of the Algorithm-MAX VALUE Table of Manpower Database.

This test takes the case of four officers to be distributed on four jobs. The algorithm runs instantly. Below are the results.

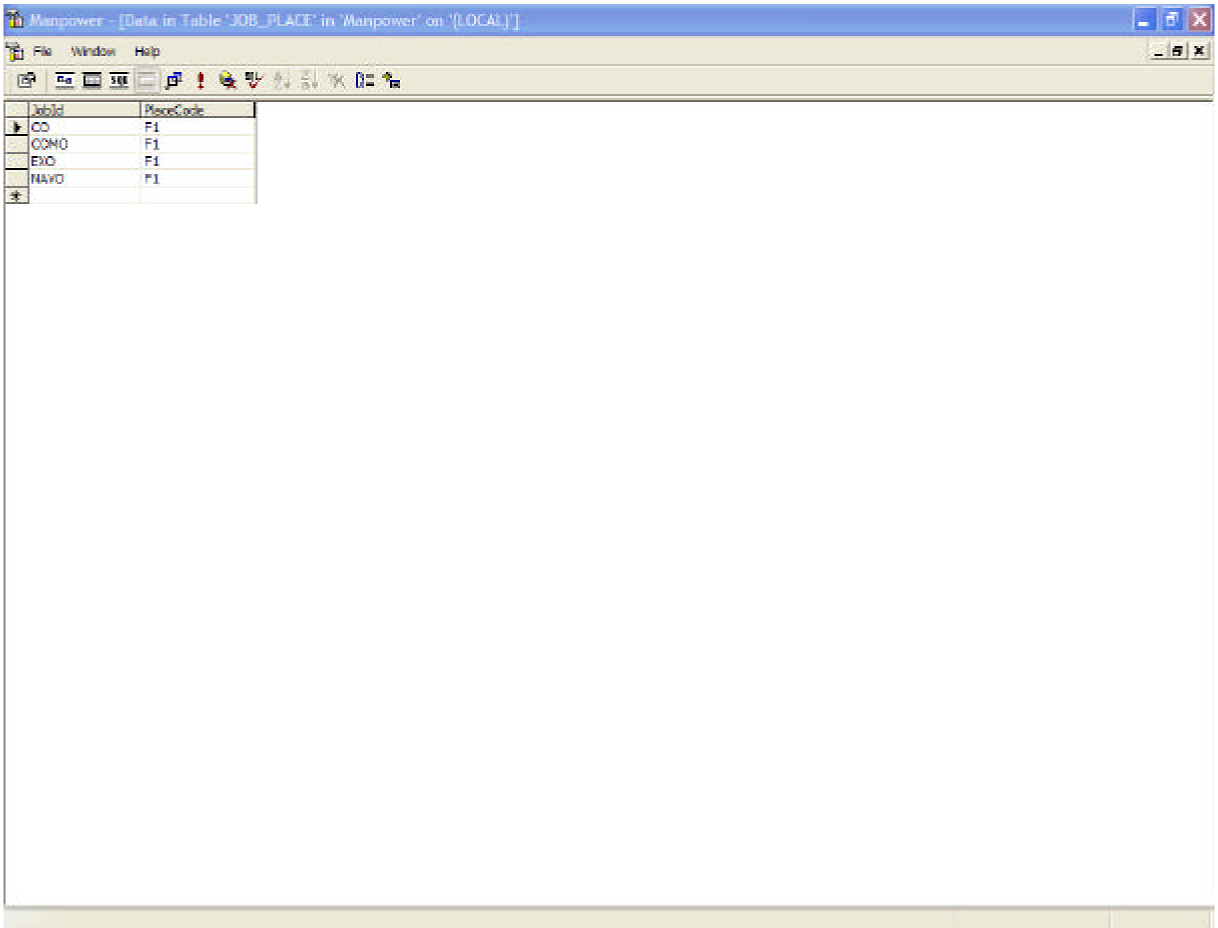
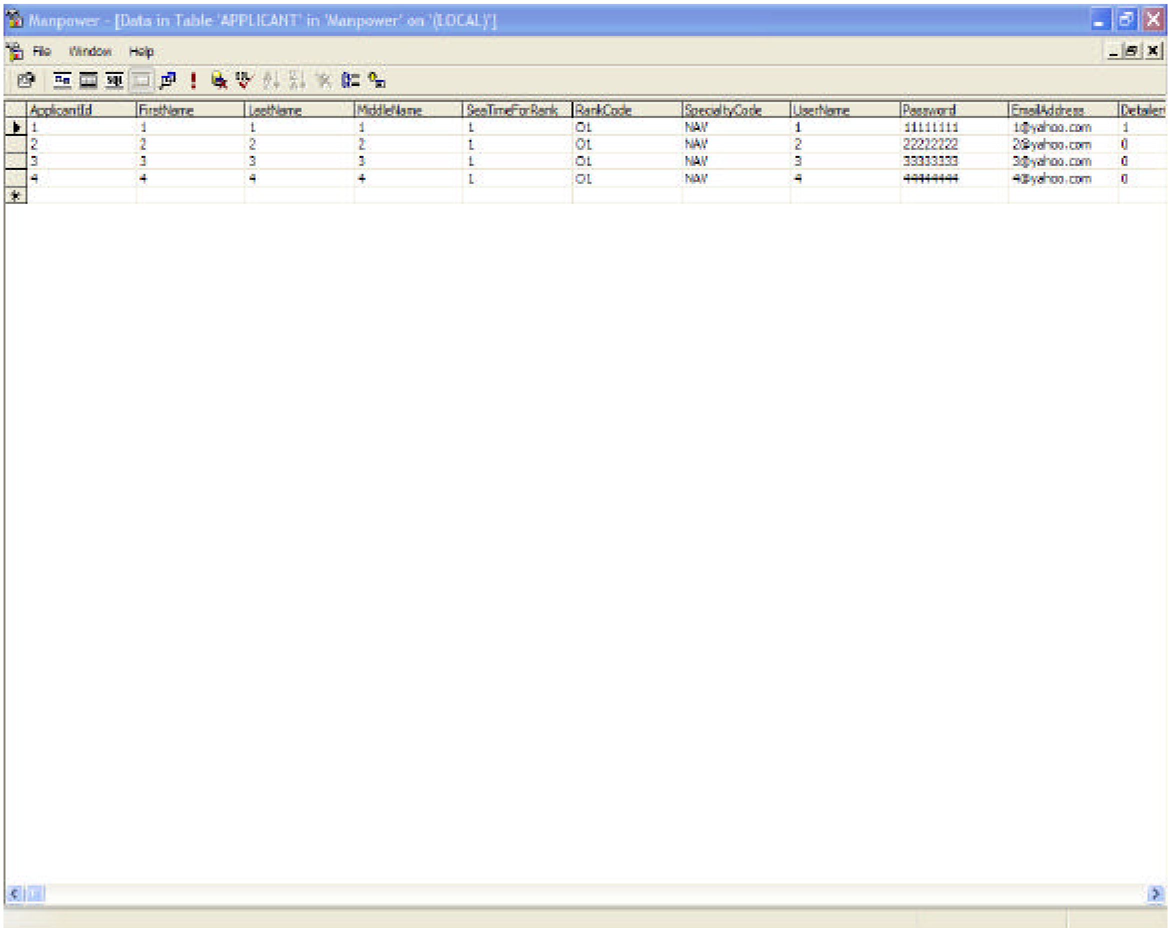
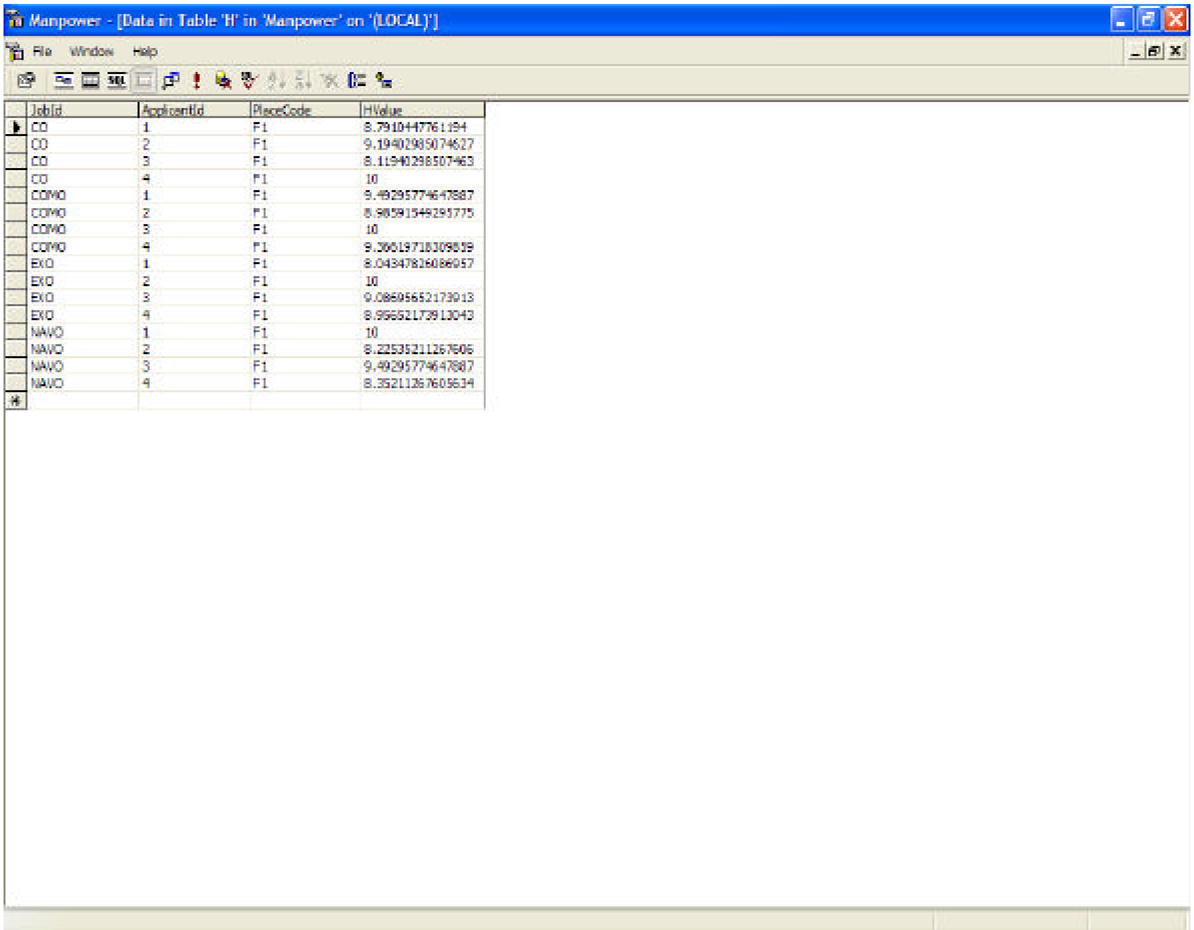


Figure 29. Job-Platform Pairs to Be Fulfilled-Manpower Database.



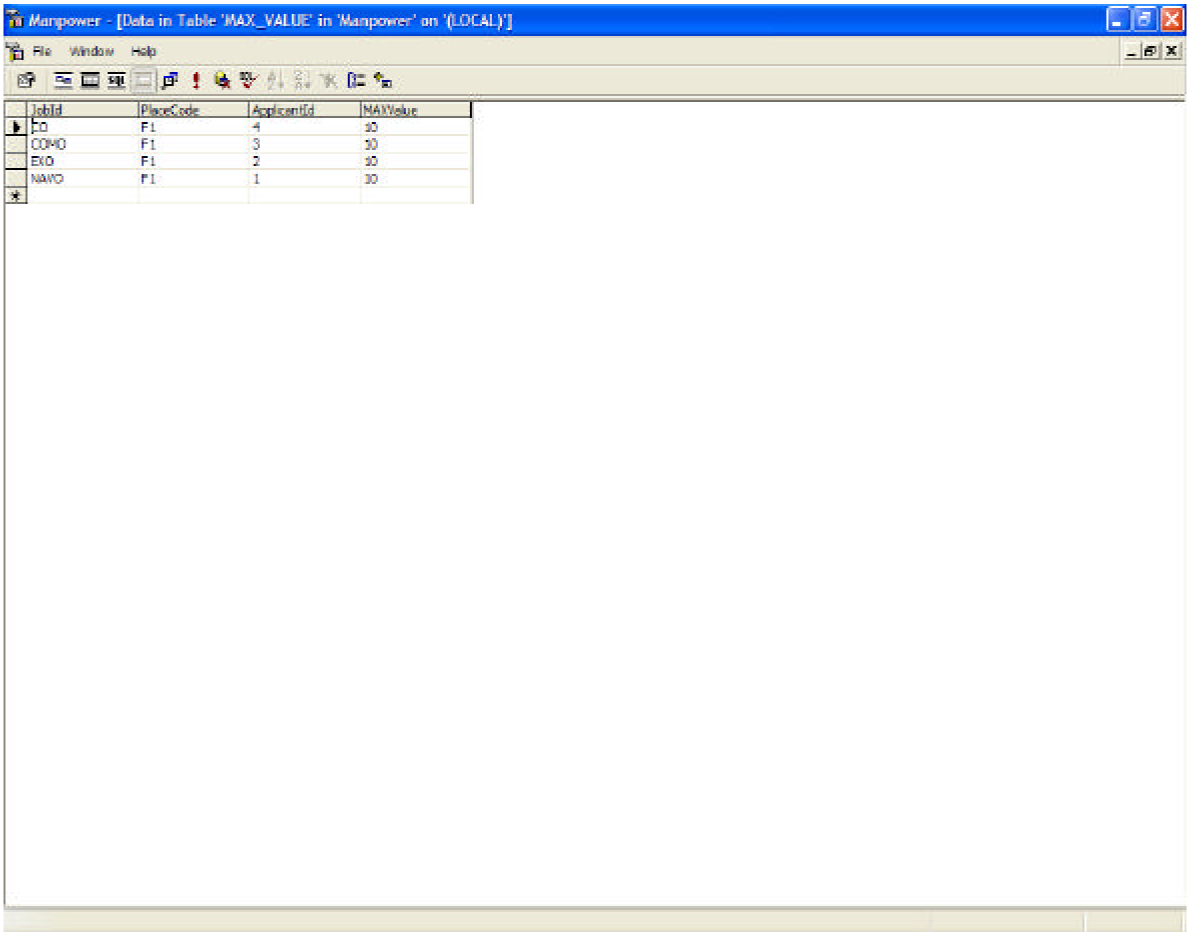
ApplicantId	FirstName	LastName	MiddleName	SeaTimeForRank	RankCode	SpecialtyCode	UserName	Password	EmailAddress	Detailer
1	1	1	1	1	O1	NAV	1	11111111	1@yahoo.com	1
2	2	2	2	1	O1	NAV	2	22222222	2@yahoo.com	0
3	3	3	3	1	O1	NAV	3	33333333	3@yahoo.com	0
4	4	4	4	1	O1	NAV	4	44444444	4@yahoo.com	0

Figure 30. Officers To Be Assigned to the Job-Platform Pairs Above-Manpower Database.



JobId	ApplicantId	PlaceCode	HValue
CO	1	F1	8.7910447761194
CO	2	F1	9.19402985074627
CO	3	F1	8.11940298507463
CO	4	F1	10
COMO	1	F1	9.49295774647887
COMO	2	F1	8.98591549295775
COMO	3	F1	10
COMO	4	F1	9.36619718309859
EXO	1	F1	8.04347826086957
EXO	2	F1	10
EXO	3	F1	9.08695652173913
EXO	4	F1	8.95652173913043
NAVO	1	F1	10
NAVO	2	F1	8.22535211267606
NAVO	3	F1	9.49295774647887
NAVO	4	F1	8.35211267605634

Figure 31. H Table-Manpower Database.



JobId	PlaceCode	ApplicantId	MAXValue
CO	F1	4	10
COMO	F1	3	10
EXO	F1	2	10
NAVO	F1	1	10

Figure 32. The Solution of the Algorithm-MAX VALUE Table of Manpower Database.

Apparently, for large loads of jobs the computational time will increase significantly. Specifically, suppose that the set of jobs is n. From the design of the algorithm the worst case computational time is $O(n^2)$. The reason is that the algorithm may backtrack until it finds a path in order to fulfill all the jobs. The worst case scenario will be that the algorithm backtracks for every officer, beginning from the highest priority job until the lowest priority job and then backtracks to highest priority job again. This means that the algorithm goes back and forth for all n officers n times, which concludes to the $O(n^2)$ computational time.

The computational time for the average case scenario is expected to be O(n), since the algorithm won't backtrack a lot. Usually, it tracks back a couple of times for a couple of jobs. So it will begin from the highest priority job and end to the lowest priority job for a total computational time of O(n).

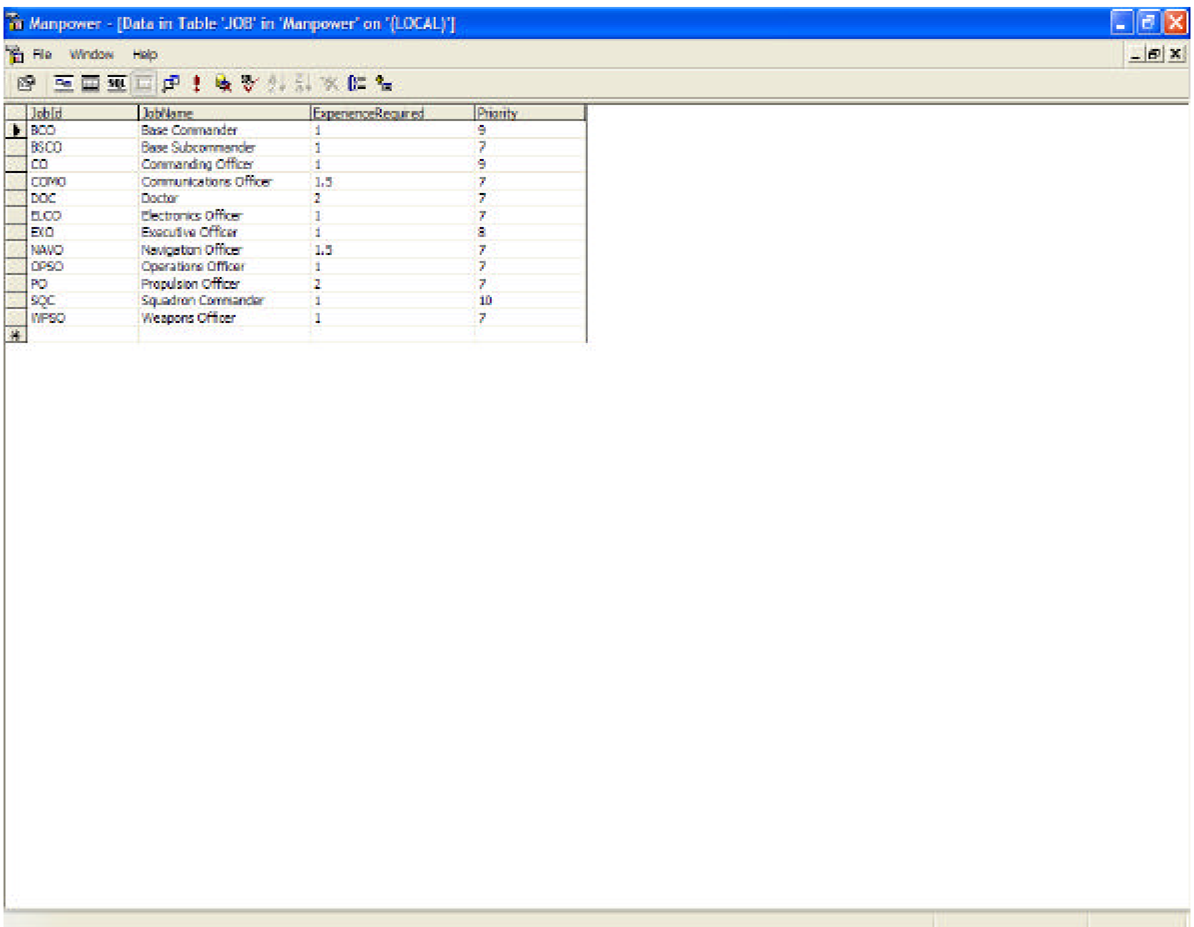
2. Increases on the Estimate Function Result When Changes Are Made on the Algorithm's Solution

In order to show the changes, the following scenario of available jobs and officers is put into the Manpower database.

APPLICANT table: The same with figure 24.

JOB PLACE table: The same with figure 23.

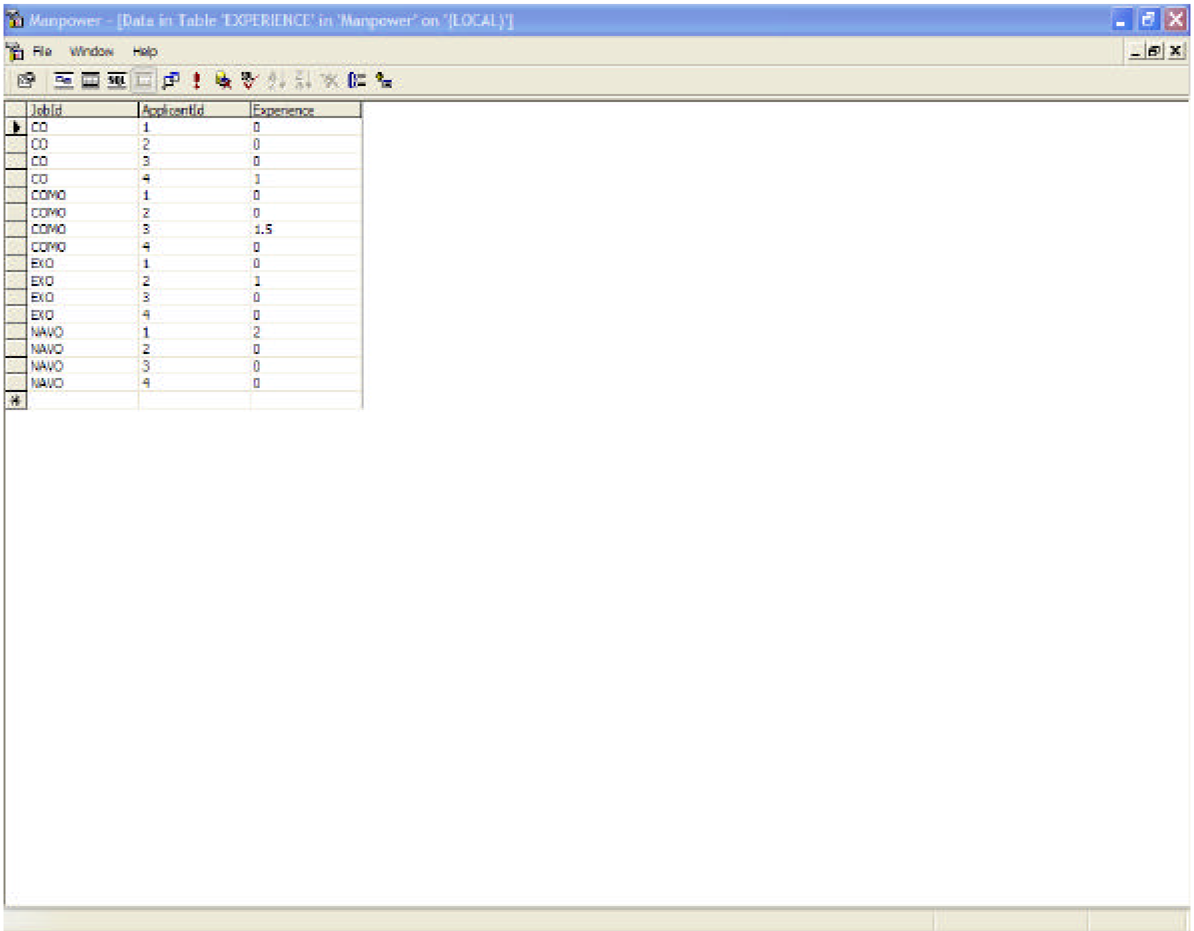
JOB table:



JobId	JobName	ExperienceRequired	Priority
BCO	Base Commander	1	9
BSCO	Base Subcommander	1	7
CO	Commanding Officer	1	9
COMO	Communications Officer	1.5	7
DOC	Doctor	2	7
ELCO	Electronics Officer	1	7
EXO	Executive Officer	1	8
NAVO	Navigation Officer	1.5	7
OPSO	Operations Officer	1	7
PO	Propulsion Officer	2	7
SQC	Squadron Commander	1	10
WPSO	Weapons Officer	1	7

Figure 33. JOB Table-Manpower Database.

EXPERIENCE table:



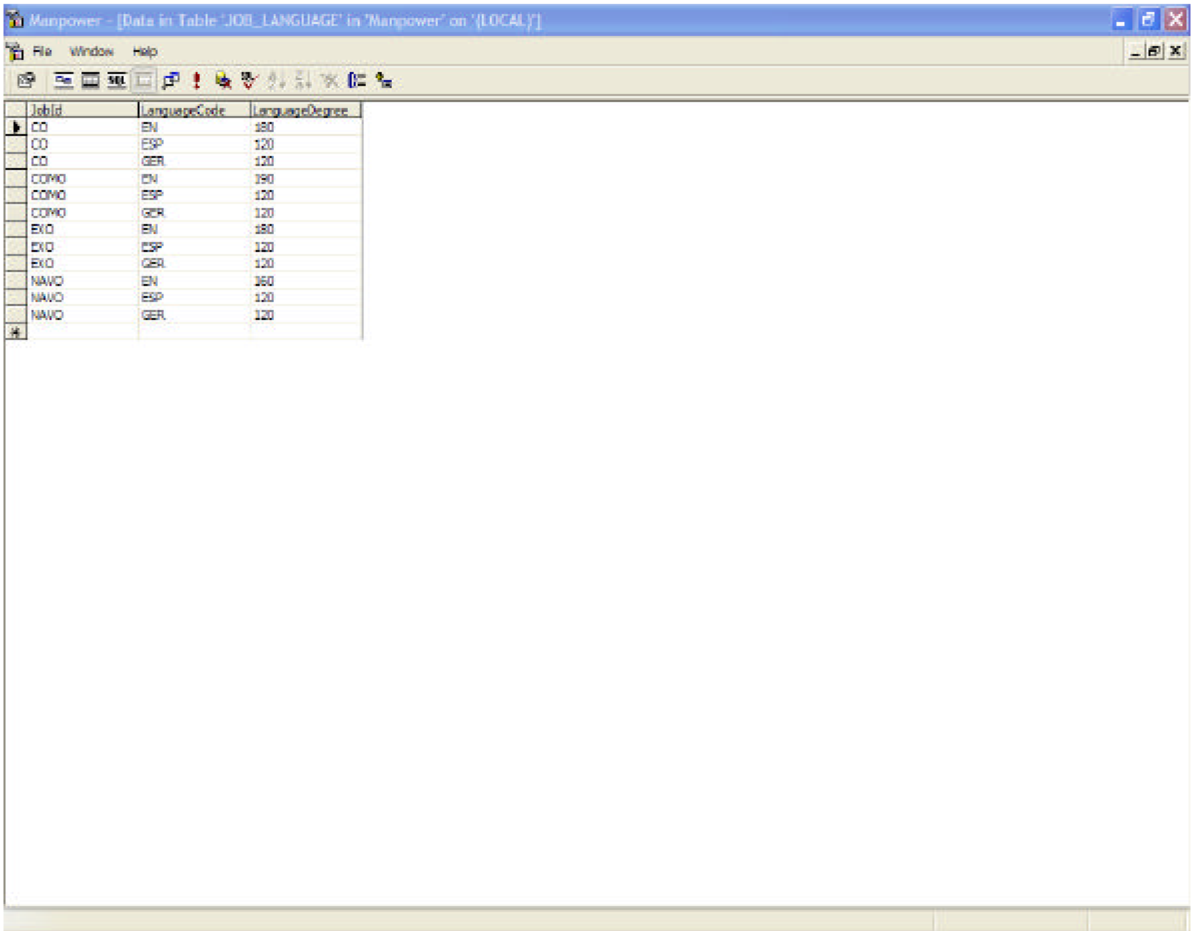
Manpower - [Data in Table 'EXPERIENCE' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	ApplicantId	Experience
CO	1	0
CO	2	0
CO	3	0
CO	4	1
COMO	1	0
COMO	2	0
COMO	3	1.5
COMO	4	0
EXO	1	0
EXO	2	1
EXO	3	0
EXO	4	0
NAVO	1	2
NAVO	2	0
NAVO	3	0
NAVO	4	0

Figure 34. EXPERIENCE Table-Manpower Database.

JOB LANGUAGE table:

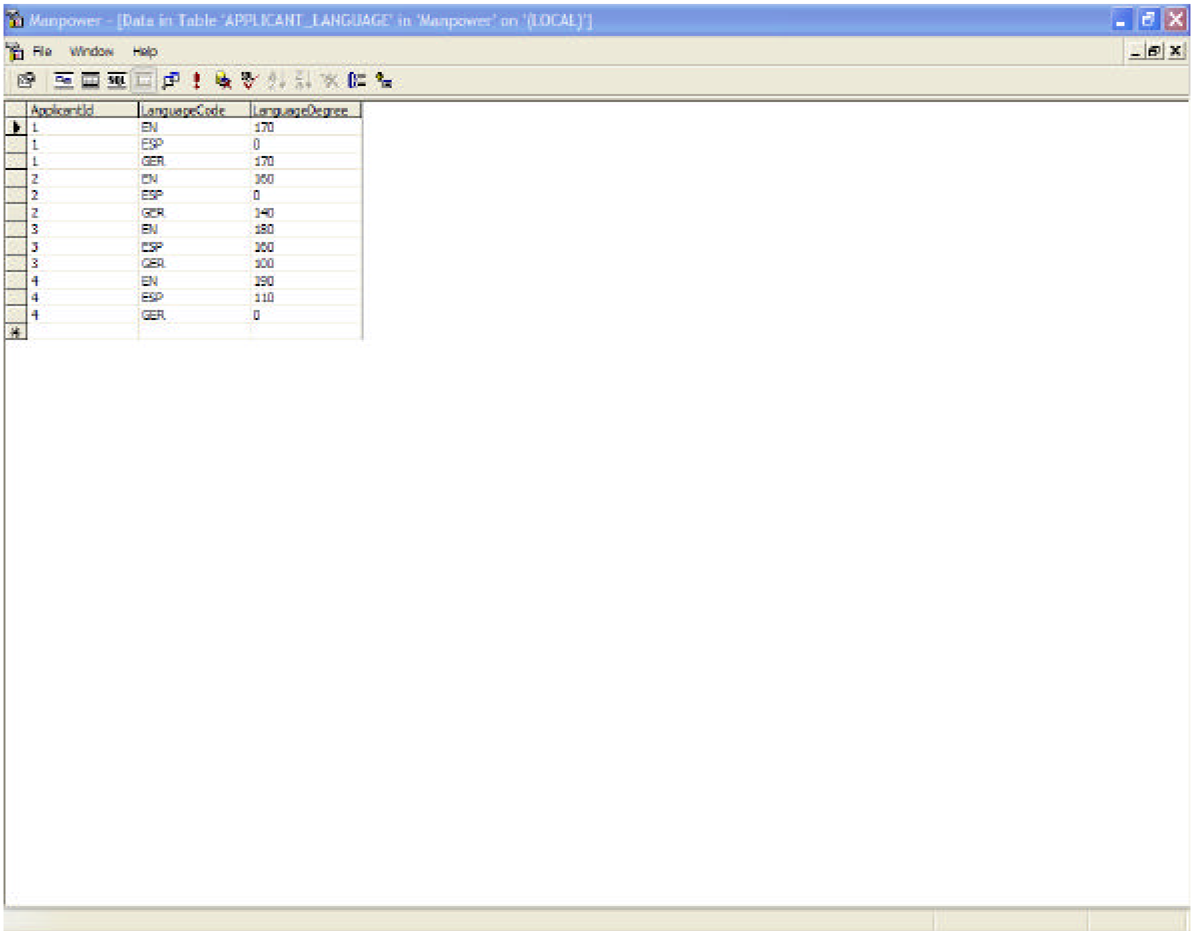

Manpower - [Data in Table 'JOB_LANGUAGE' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	LanguageCode	LanguageDegree
CO	EN	180
CO	ESP	120
CO	GER	120
COMO	EN	190
COMO	ESP	120
COMO	GER	120
EXO	EN	180
EXO	ESP	120
EXO	GER	120
NAVO	EN	160
NAVO	ESP	120
NAVO	GER	120

Figure 35. JOB LANGUAGE Table-Manpower Database.

APPLICANT LANGUAGE table:

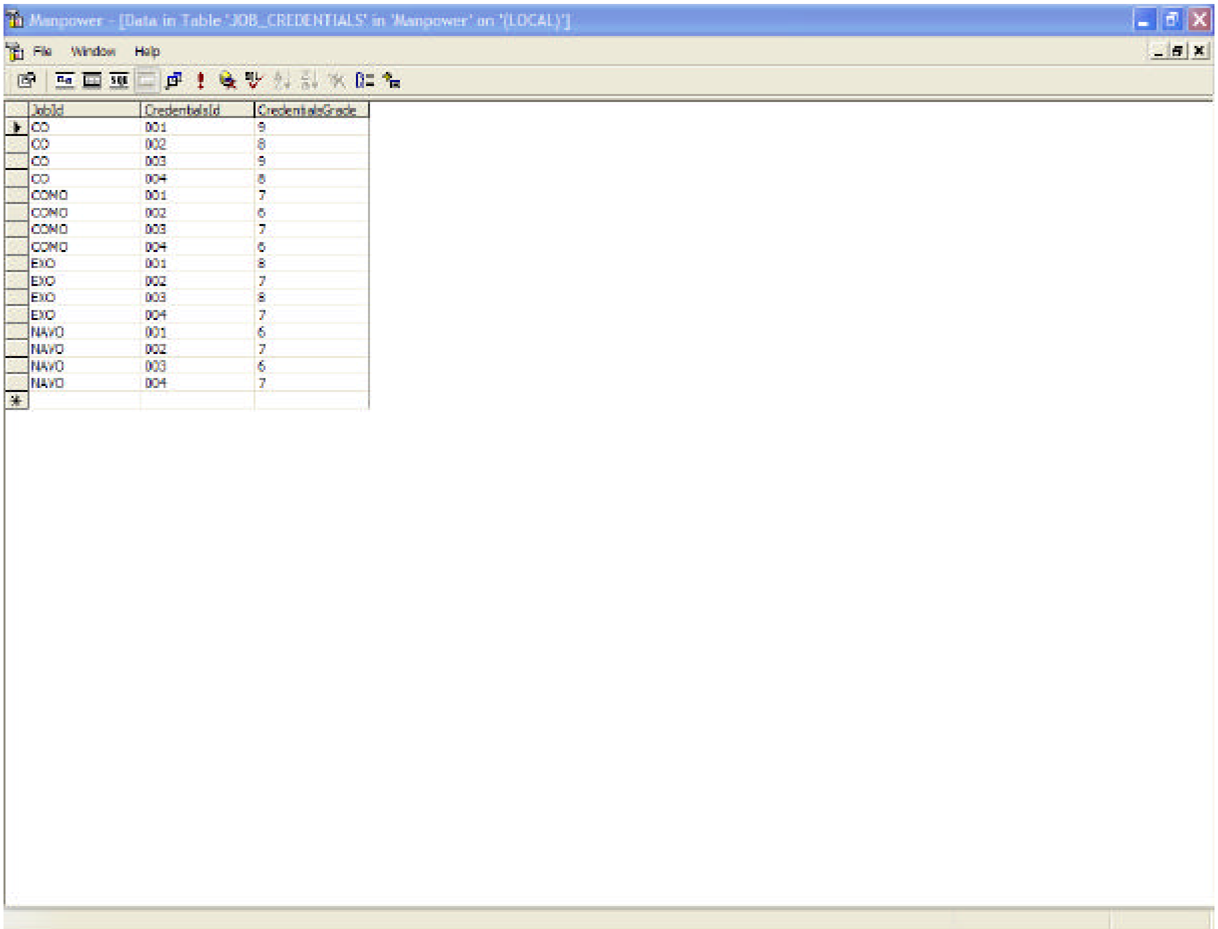

Manpower - [Data in Table 'APPLICANT_LANGUAGE' in 'Manpower' on '(LOCAL)']

File Window Help

ApplicantId	LanguageCode	LanguageDegree
1	EN	170
1	ESP	0
1	GER	170
2	EN	160
2	ESP	0
2	GER	140
3	EN	180
3	ESP	160
3	GER	100
4	EN	190
4	ESP	110
4	GER	0

Figure 36. APPLICANT LANGUAGE Table-Manpower Database.

JOB CREDENTIALS table:



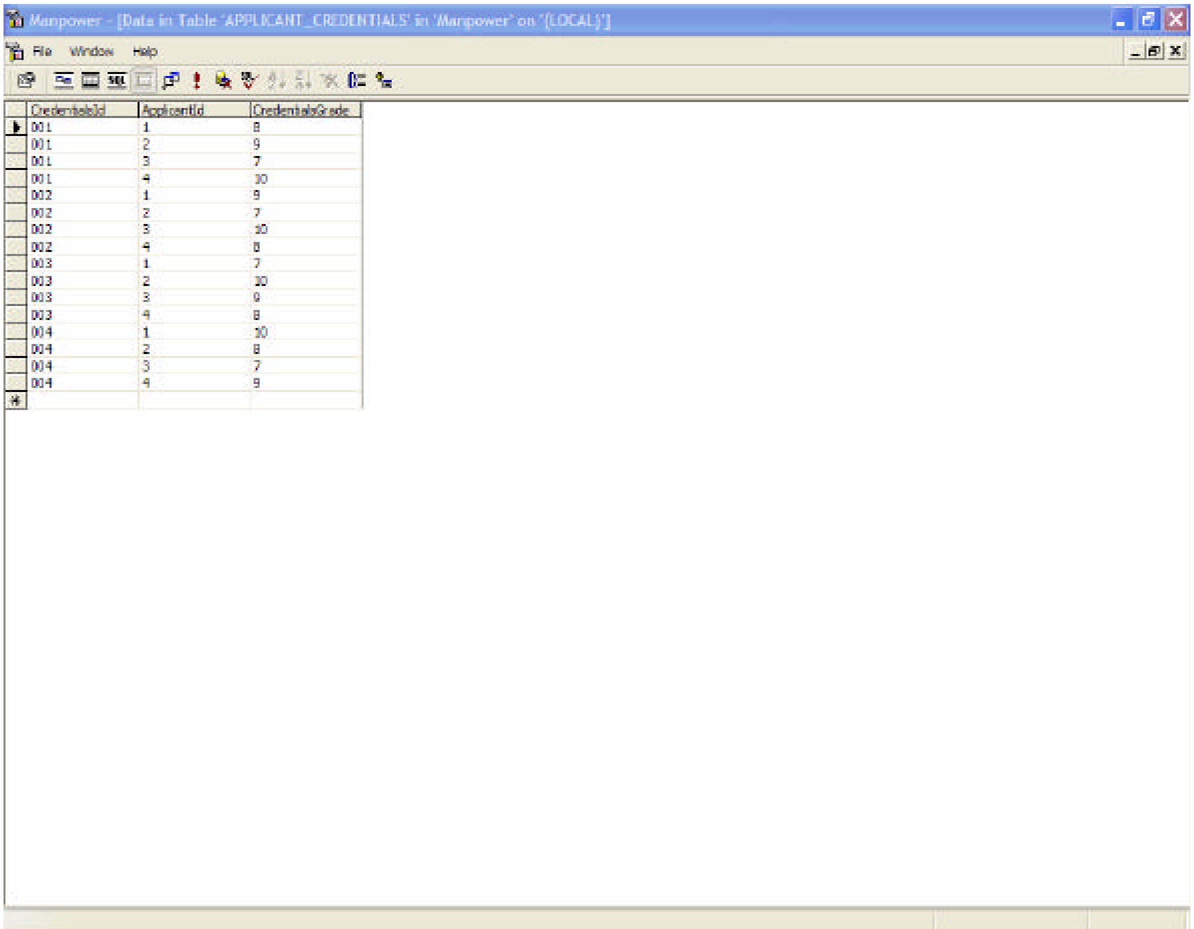
Manpower - [Data in Table 'JOB_CREDENTIALS' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	CredentialsId	CredentialsGrade
CO	001	9
CO	002	8
CO	003	9
CO	004	8
CONO	001	7
CONO	002	6
CONO	003	7
CONO	004	6
EXO	001	8
EXO	002	7
EXO	003	8
EXO	004	7
NAVO	001	6
NAVO	002	7
NAVO	003	6
NAVO	004	7

Figure 37. JOB CREDENTIALS Table-Manpower Database.

APPLICANT CREDENTIALS table:



CredentialsId	ApplicantId	CredentialsGrade
001	1	8
001	2	9
001	3	7
001	4	10
002	1	9
002	2	7
002	3	10
002	4	8
003	1	7
003	2	10
003	3	9
003	4	8
004	1	10
004	2	8
004	3	7
004	4	9

Figure 38. APPLICANT CREDENTIALS Table-Manpower Database.

JOB QUALIFICATION table:

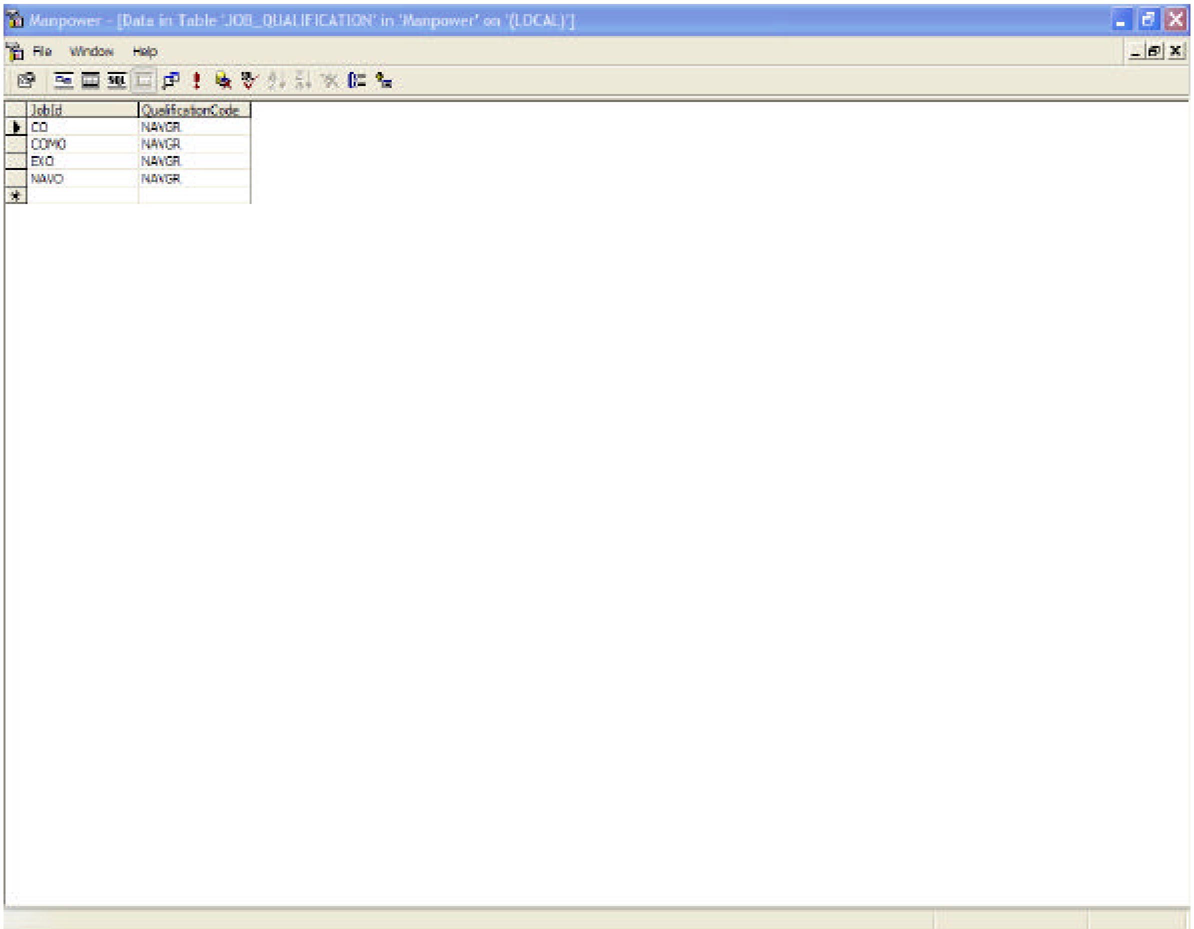


Figure 39. JOB QUALIFICATION Table-Manpower Database.

QUALIFICATION APPLICANT table:

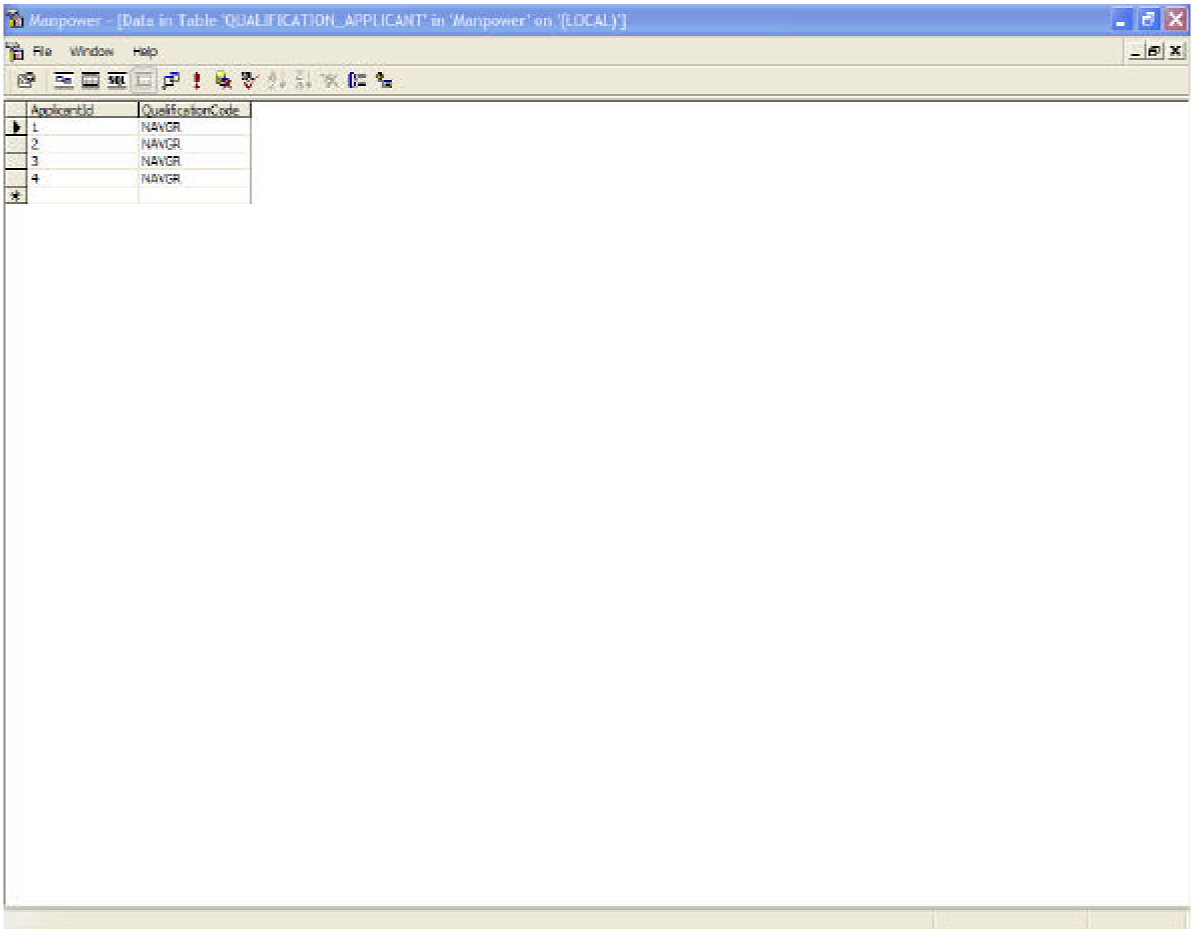
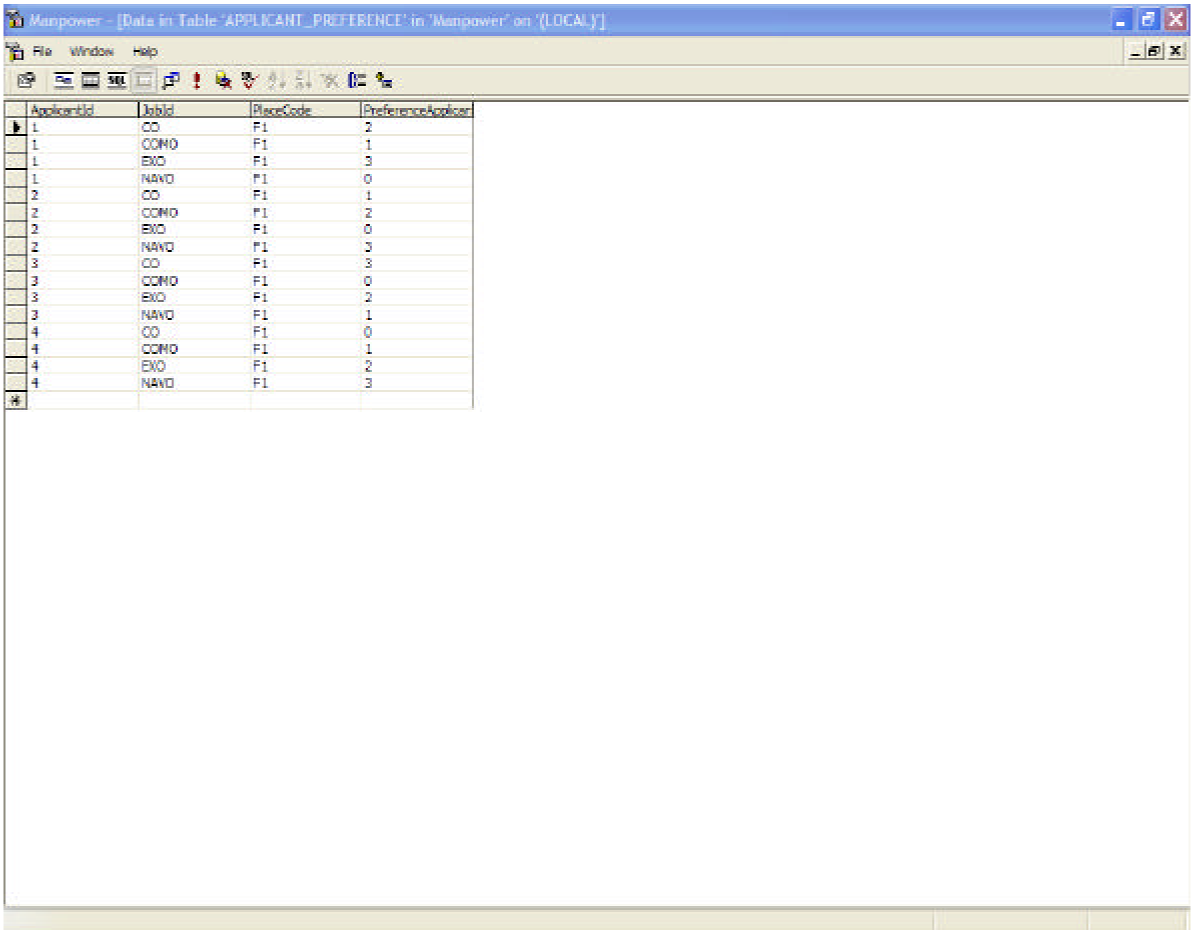


Figure 40. QUALIFICATION APPLICANT Table-Manpower Database.

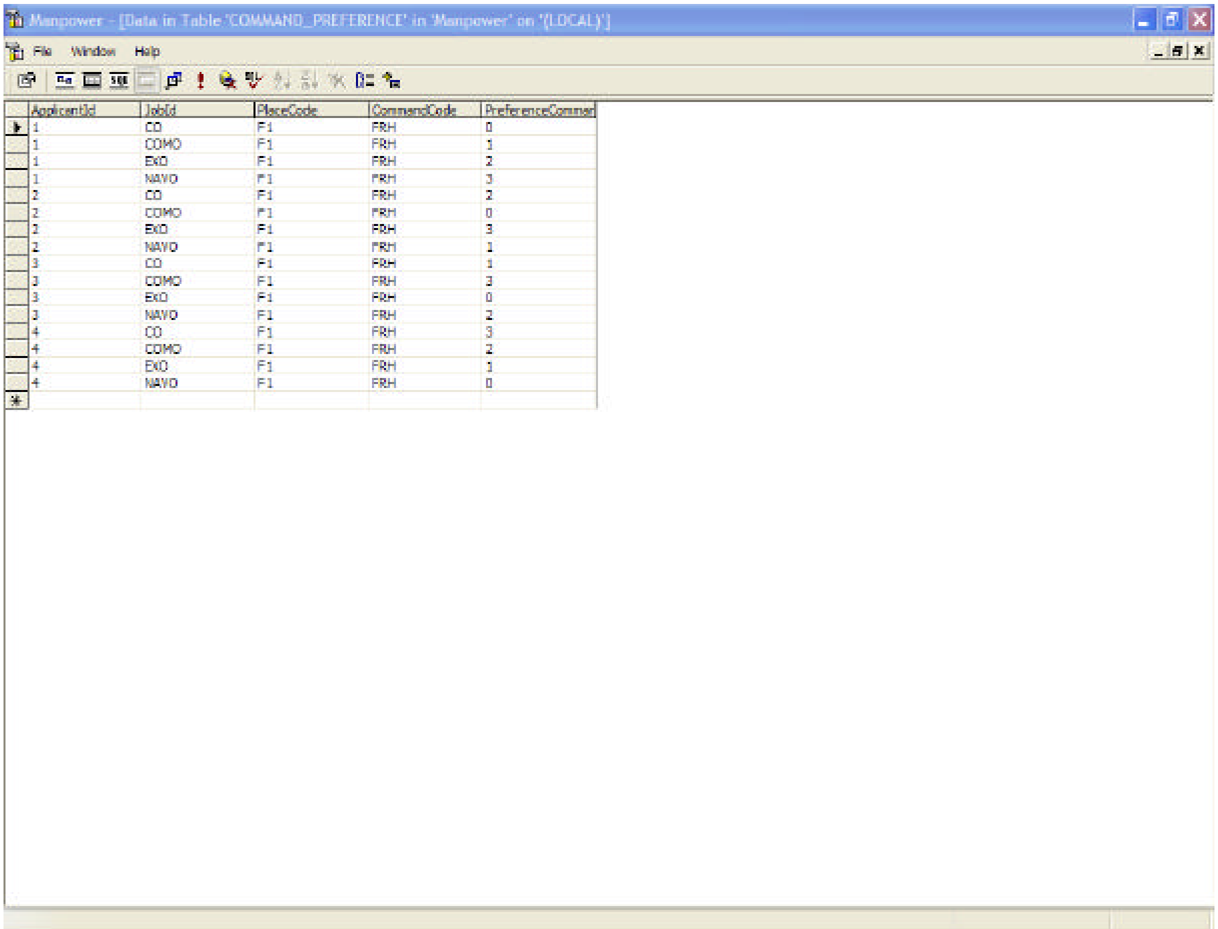
APPLICANT PREFERENCE table:



ApplicantId	JobId	PlaceCode	PreferenceApplicar
1	CO	F1	2
1	COMO	F1	1
1	EXO	F1	3
1	NAVO	F1	0
2	CO	F1	1
2	COMO	F1	2
2	EXO	F1	0
2	NAVO	F1	3
3	CO	F1	3
3	COMO	F1	0
3	EXO	F1	2
3	NAVO	F1	1
4	CO	F1	0
4	COMO	F1	1
4	EXO	F1	2
4	NAVO	F1	3

Figure 41. APPLICANT PREFERENCE Table-Manpower Database.

COMMAND PREFERENCE table:



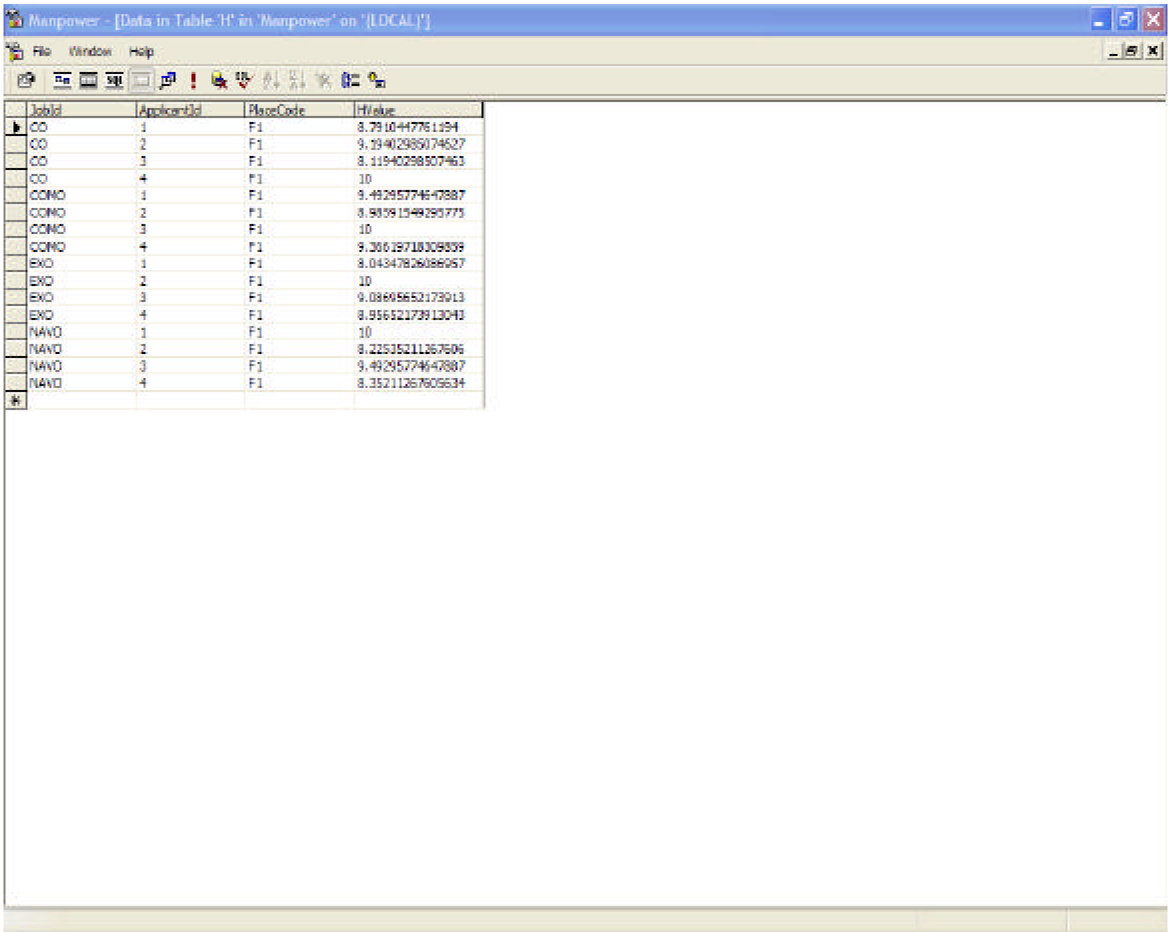
Manpower - [Data in Table 'COMMAND_PREFERENCE' in 'Manpower' on '(LOCAL)']

File Window Help

ApplicantId	JobId	PlaceCode	CommandCode	PreferenceCommand
1	CO	F1	FRH	0
1	COMO	F1	FRH	1
1	EXO	F1	FRH	2
1	NAVO	F1	FRH	3
2	CO	F1	FRH	2
2	COMO	F1	FRH	0
2	EXO	F1	FRH	3
2	NAVO	F1	FRH	1
3	CO	F1	FRH	1
3	COMO	F1	FRH	3
3	EXO	F1	FRH	0
3	NAVO	F1	FRH	2
4	CO	F1	FRH	3
4	COMO	F1	FRH	2
4	EXO	F1	FRH	1
4	NAVO	F1	FRH	0

Figure 42. COMMAND PREFERENCE Table-Manpower Database.

After the algorithm is ran, the H table becomes as shown in the figure below.



JobId	ApplicantId	PlaceCode	HValue
CO	1	F1	8.7910447761194
CO	2	F1	9.19402985074627
CO	3	F1	8.11940298507463
CO	4	F1	10
COMO	1	F1	9.49295774647887
COMO	2	F1	8.98591549295775
COMO	3	F1	10
COMO	4	F1	9.36619718309859
EXO	1	F1	8.04347826086957
EXO	2	F1	10
EXO	3	F1	9.08695652173913
EXO	4	F1	8.95652173913043
NAVO	1	F1	10
NAVO	2	F1	8.22535211267606
NAVO	3	F1	9.49295774647887
NAVO	4	F1	8.35211267605634

Figure 43. H Table-Manpower Database.

The MAX Value table results are shown in the figure below.

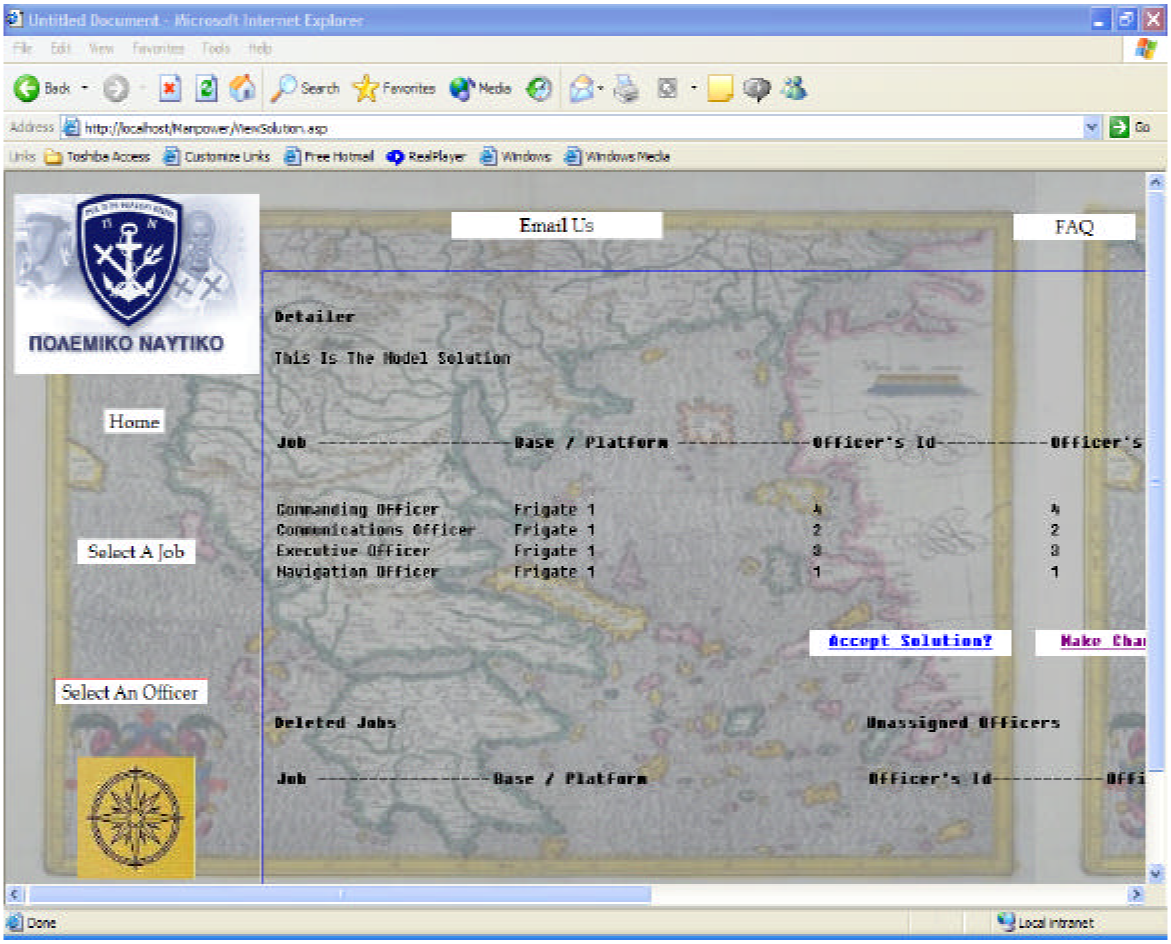


Figure 44. Solution (Screen 1)-Manpower Database.

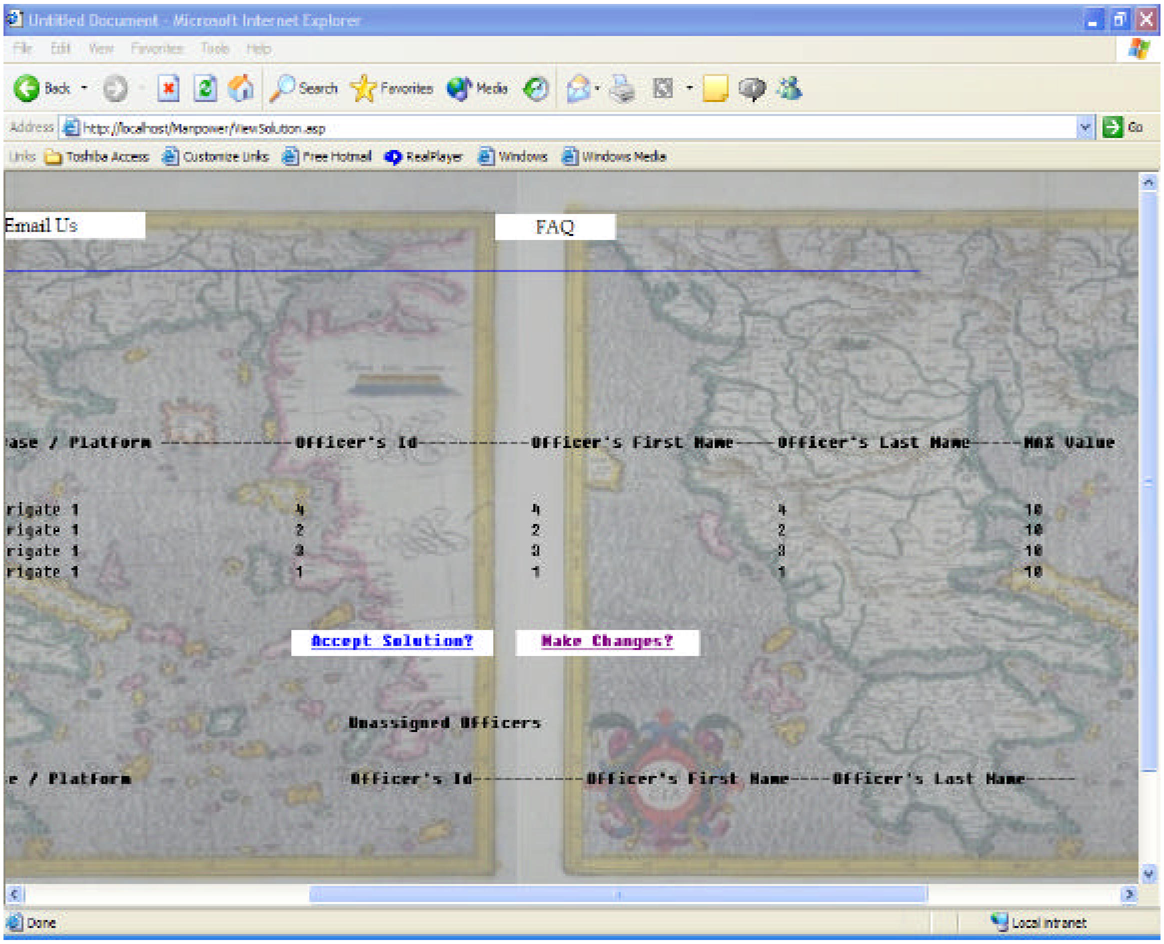


Figure 45. Solution (Screen 2)-Manpower Database.

The detailer then makes the following change. He/she assigns the Commanding Officer's job of the ship Frigate 1 to the officer 3, and the Executive Officer's job of the ship Frigate 1 to the officer 4.

The following screenshots show the new results on the solution and the Estimate Function.

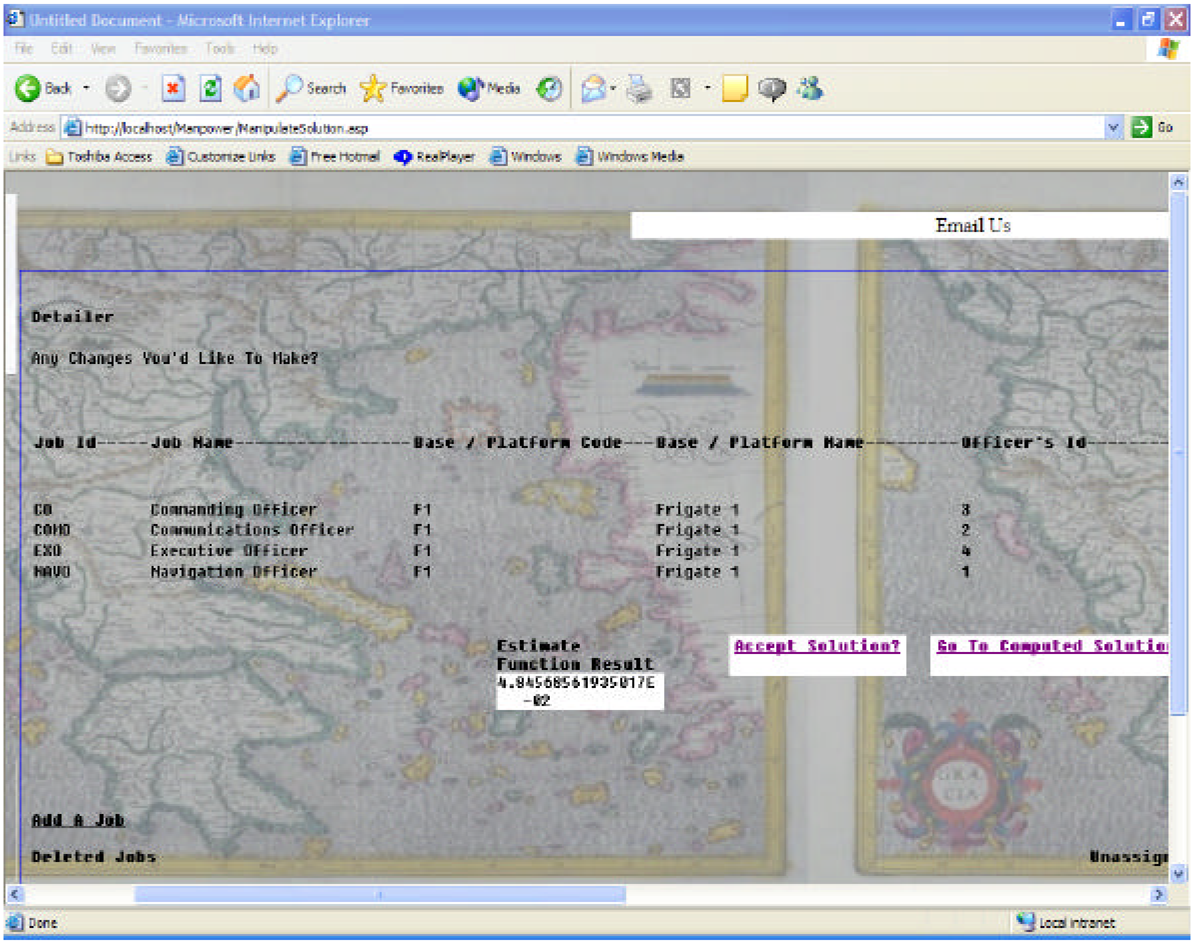


Figure 46. Change on the Solution and Estimate Function (Screen 1)-Manpower Database.

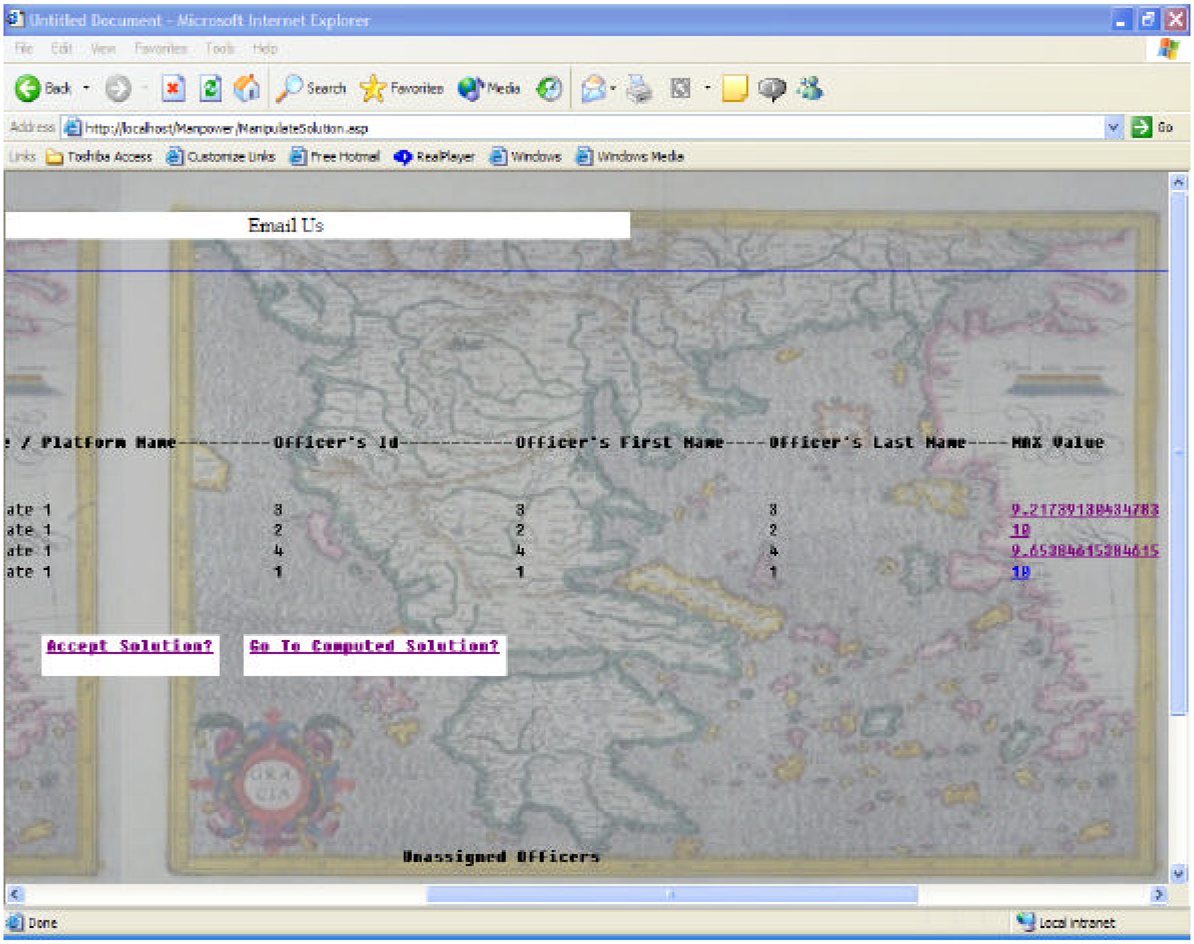


Figure 47. Change on the Solution and Estimate Function (Screen 2)-Manpower Database.

Apparently, the detailer selected officers with worse HValues than the algorithm selected. This resulted in an increase of the Estimate Function by 0.0485 units.

3. Changes on the Algorithm's Distribution, When Different Coefficient Weights for the Decision Variables Are Given

For the case just described above, the solution of the algorithm presented in Figures 37 and 38 was made with coefficient weights equal to 1 for all the criteria as shown in the figure below.

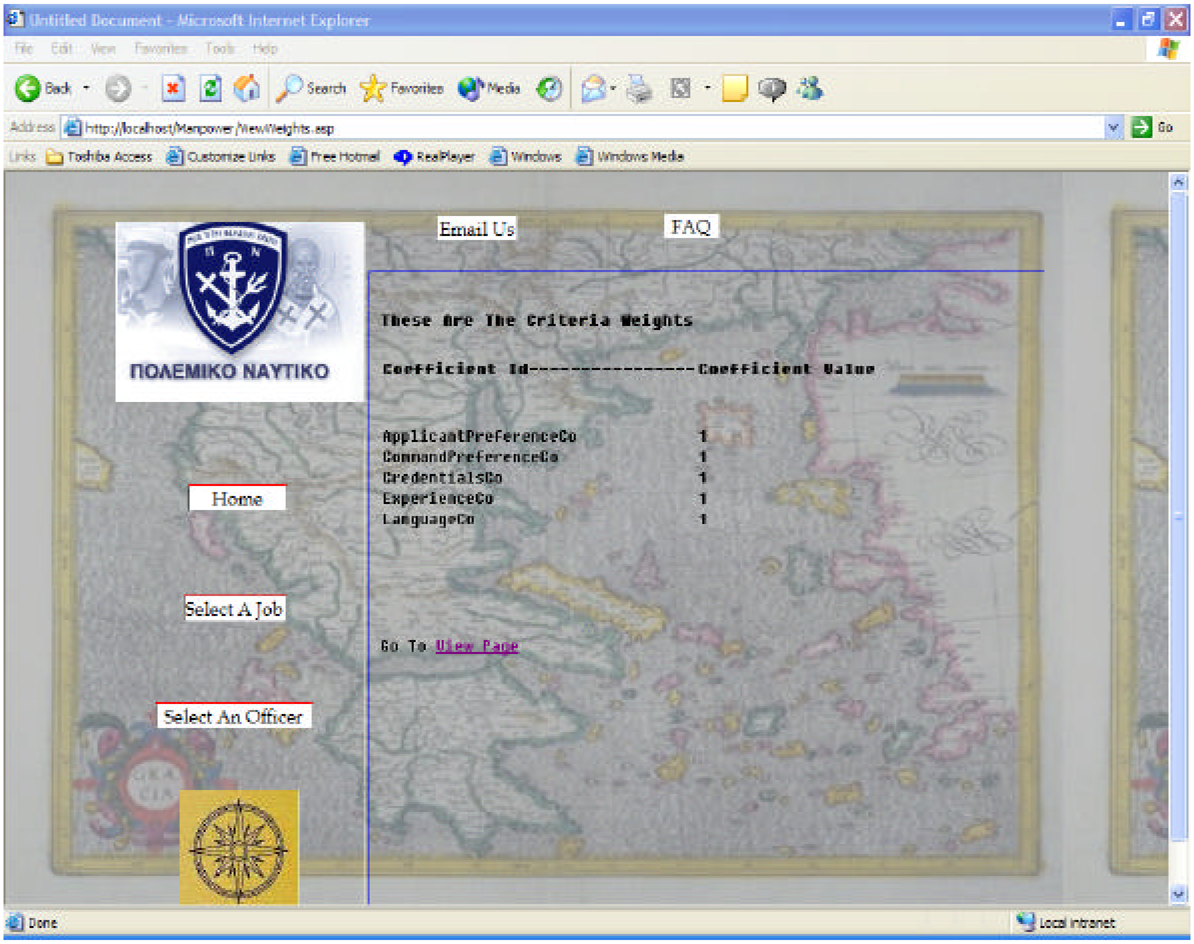


Figure 48. Coefficient Weights Per Criterion-Manpower Database.

If the detailer changes the criteria weights, both the H table and the solution change. Assume that the detailer would like to give more weight to the officers' preference and the commands' preference than to their Credentials, Experience and Language criteria. He/she decides then to put weight 5 to the officers' and commands' preference criteria and leave the rest criteria as they are.

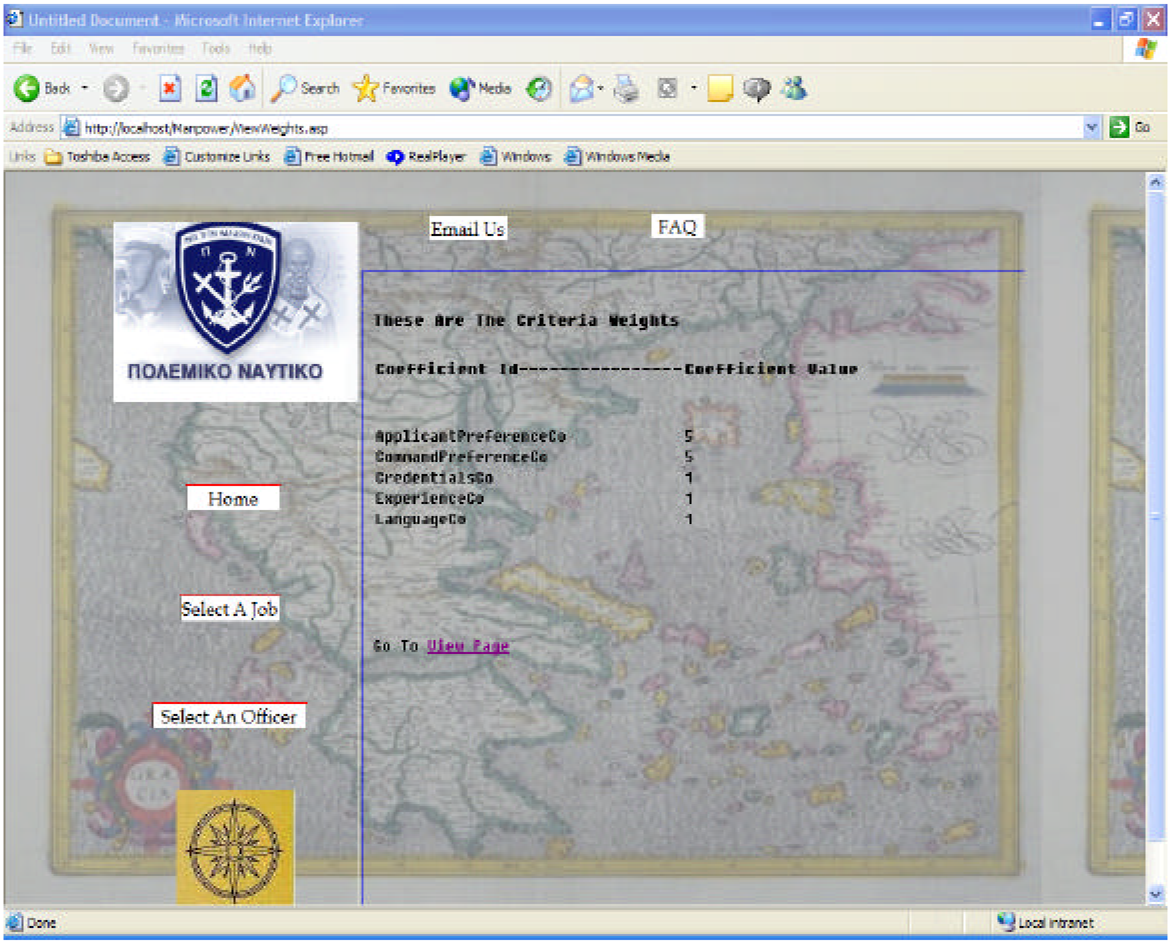
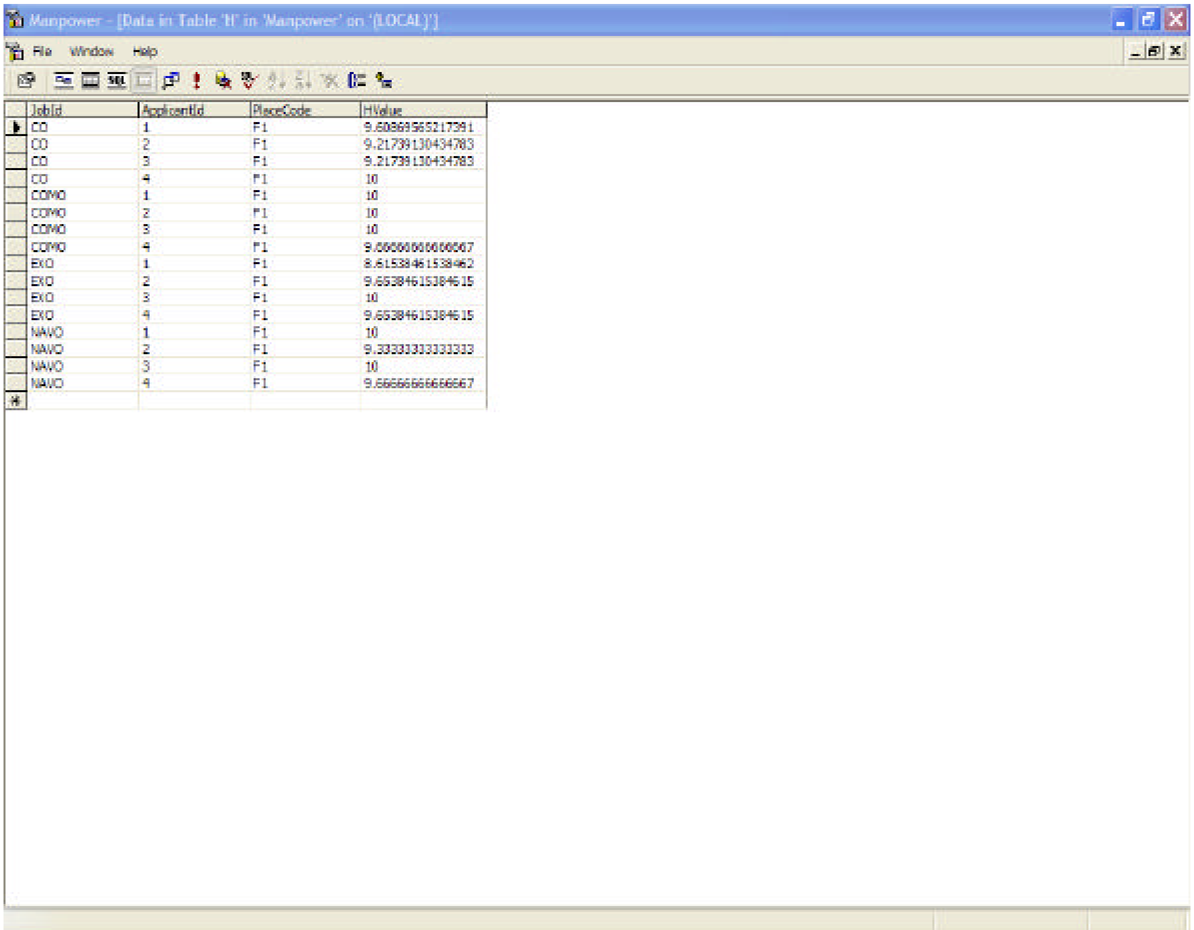


Figure 49. Coefficient Weights Per Criterion After the Weights Change-Manpower Database.

Now that the weights are changed, a different H table and a different solution will be produced. The two figures below show that change on the H table.

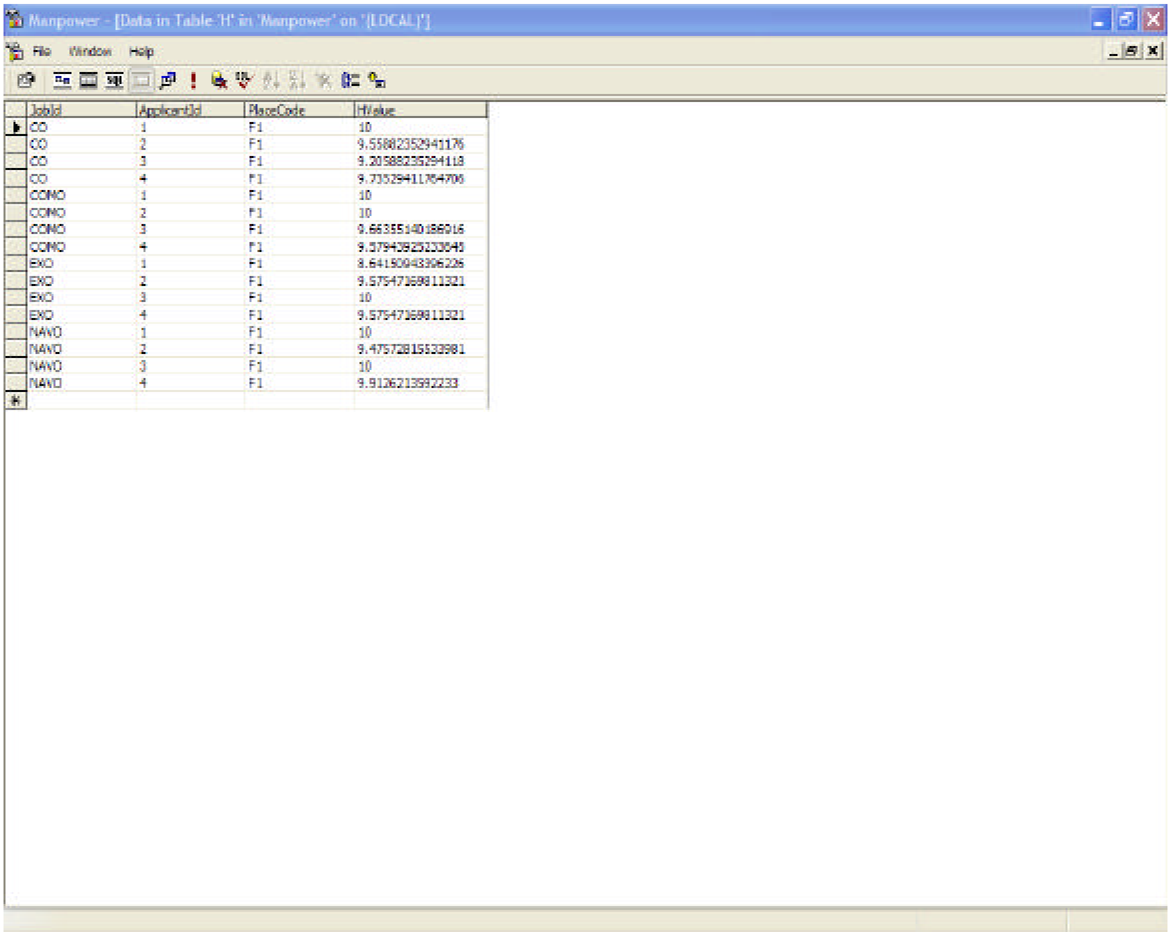


Manpower - [Data in Table 'H' in 'Manpower' on '(LOCAL)']

File Window Help

JobId	ApplicantId	PlaceCode	HValue
CO	1	F1	9.60869565217391
CO	2	F1	9.21739130434783
CO	3	F1	9.21739130434783
CO	4	F1	10
COMO	1	F1	10
COMO	2	F1	10
COMO	3	F1	10
COMO	4	F1	9.66666666666667
EXO	1	F1	8.61538461538462
EXO	2	F1	9.65384615384615
EXO	3	F1	10
EXO	4	F1	9.65384615384615
NAVO	1	F1	10
NAVO	2	F1	9.33333333333333
NAVO	3	F1	10
NAVO	4	F1	9.66666666666667

Figure 50. H Table Before the Weights Change and the Algorithm Runs-Manpower Database.



JobId	ApplicantId	PlaceCode	HValue
CO	1	F1	10
CO	2	F1	9.55882352941176
CO	3	F1	9.20588235294118
CO	4	F1	9.73529411764706
COMO	1	F1	10
COMO	2	F1	10
COMO	3	F1	9.66355140186916
COMO	4	F1	9.57943925233645
EXO	1	F1	8.64150943396226
EXO	2	F1	9.57547169811321
EXO	3	F1	10
EXO	4	F1	9.57547169811321
NAVO	1	F1	10
NAVO	2	F1	9.47572815533981
NAVO	3	F1	10
NAVO	4	F1	9.9126213592233

Figure 51. H Table After the Weights Change and the Algorithm Runs-Manpower Database.

In the two figures below, the new solution of the algorithm is shown.

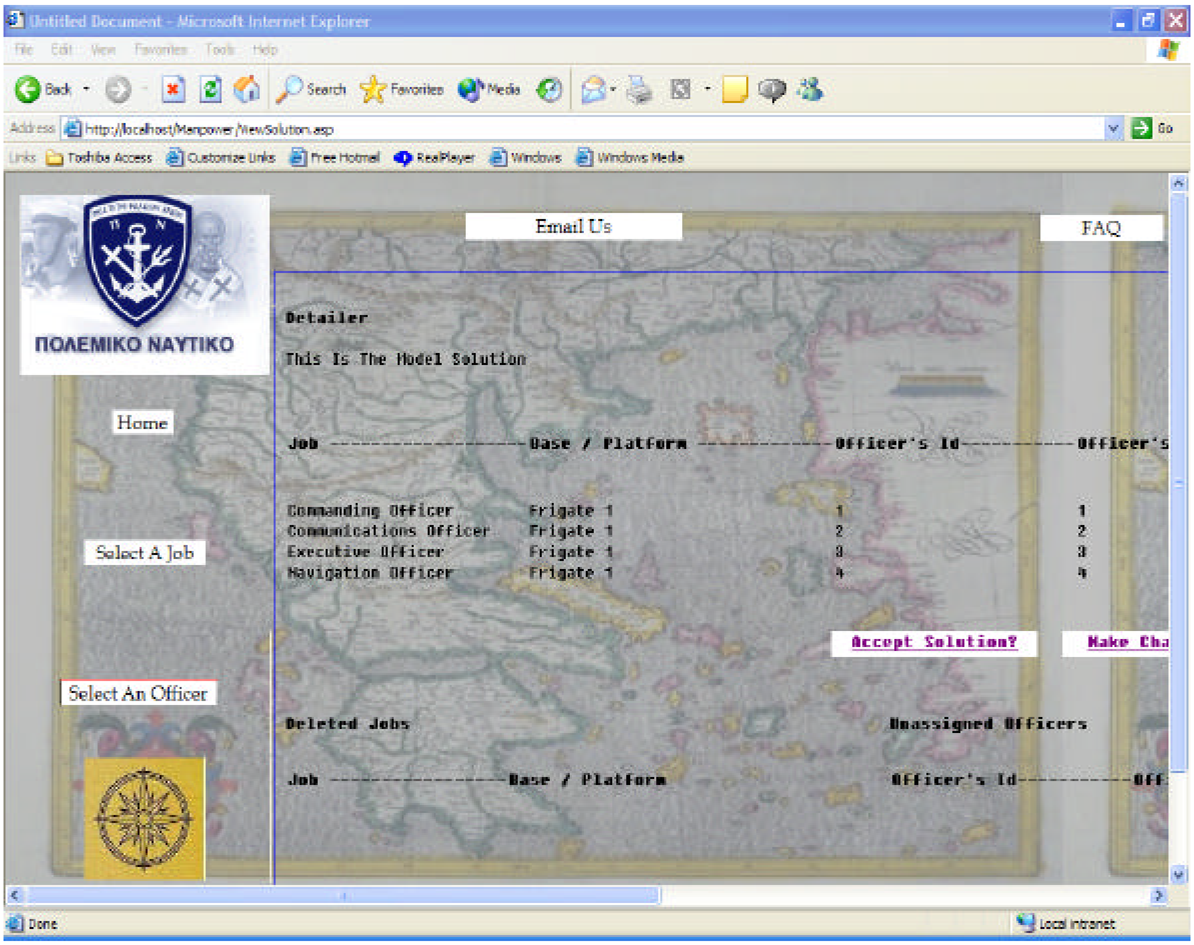


Figure 52. Solution (Screen 1)-Manpower Database.

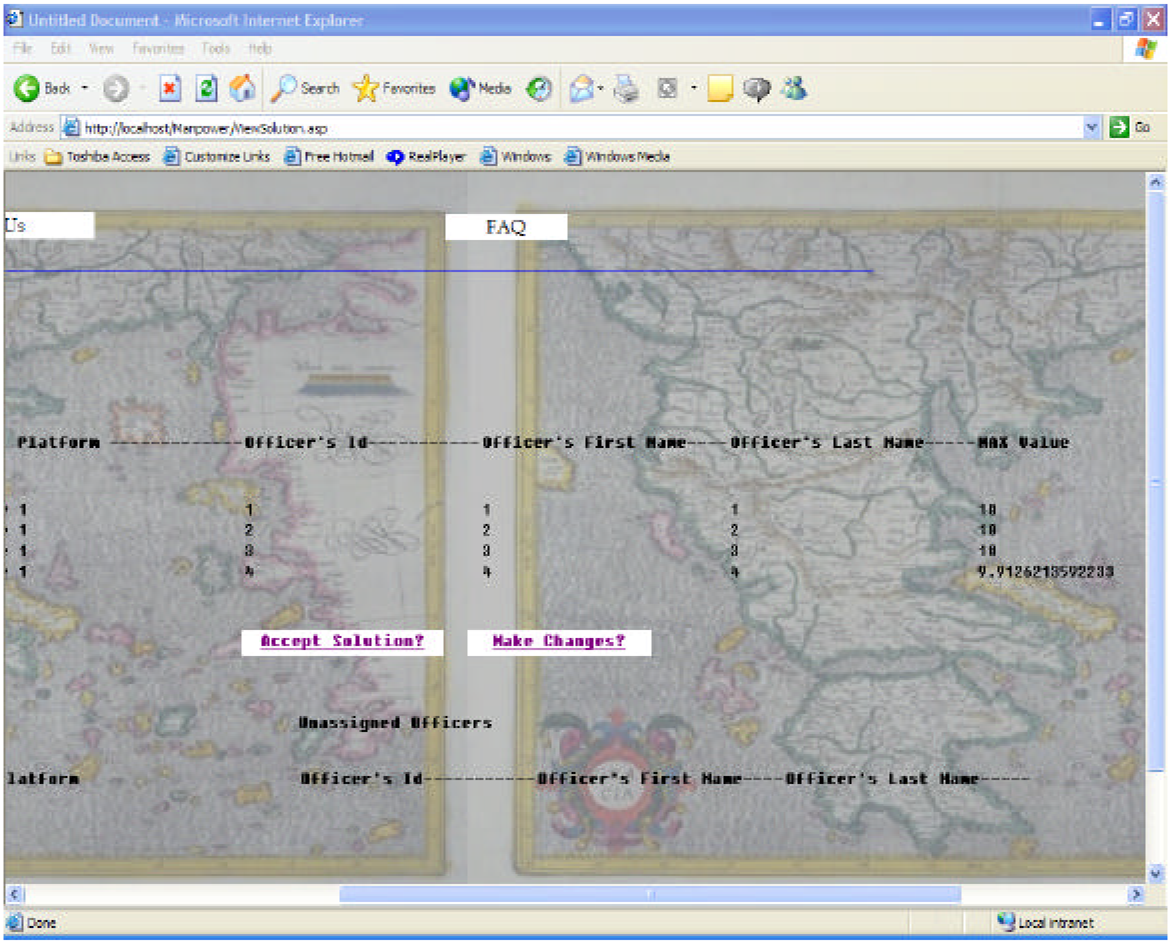


Figure 53. Solution (Screen 2)-Manpower Database.

Again, below are the 2 H tables and highlighted is the algorithm's choice of HValues for both cases.

H table before:

	CO	EXO	COMO	NAVO
1	9.608	8.615	10	**10**
2	9.217	9.653	**10**	9.333
3	9.217	**10**	10	10
4	**10**	9.653	9.666	9.666

H table after:

	CO	**EXO**	**COMO**	**NAVO**
1	**10**	8.641	10	10
2	9.558	9.575	**10**	9.475
3	9.205	**10**	9.663	10
4	9.735	9.575	9.579	**9.912**

Until now, both the Manpower database and the multi-criteria decision model are described. What remains is the description of the user interface that helps the users, the officers, the commands and the detailer to access the database and manipulate data. The next chapter describes the Manpower web site's form and structure.

THIS PAGE INTENTIONALLY LEFT BLANK

V. WEBSITE

The previous sections described the database and the multi-criteria decision tool for the Greek Navy's Manpower model. This chapter discusses the website, which helps the officers and the commands to specify their preferences and the detailer to administer the database and make decisions by using the decision support environment.

A. 3-TIER ARCHITECTURE

Before discussing the web site structure and design, it is useful to describe the 3-tier architecture model used for the implementation of this project. The figure below describes the basic form of a 3-tier architecture. The 3-tier architecture logically separates the functions of an application into a user interface component, a server business logic component, and a database component.

Many application server products and middleware products provide support for building and deploying applications using the 3-tier architecture. In most of these cases a primary role of the middle tier business logic components is to manipulate data stored in and accessed from the 3rd tier.

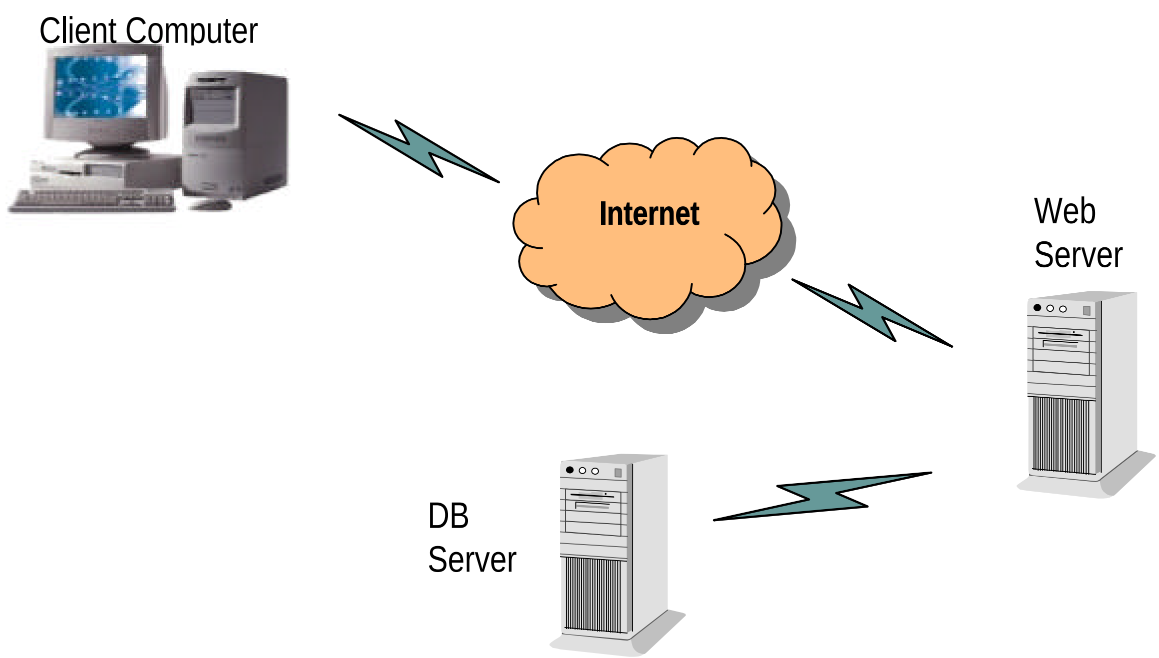


Figure 54. 3-Tier Architecture.

For this thesis, the middle-tier component is a web server running Windows IIS 5.0. The third-tier component is the Windows SQL Server 2000, which is the database server. This is the place where the data and the stored procedures of the multi-criteria decision tool reside, as described in the previous chapters. The first-tier component is the browser for the Manpower database users. The figure below describes the 3-tier architecture for our prototype.

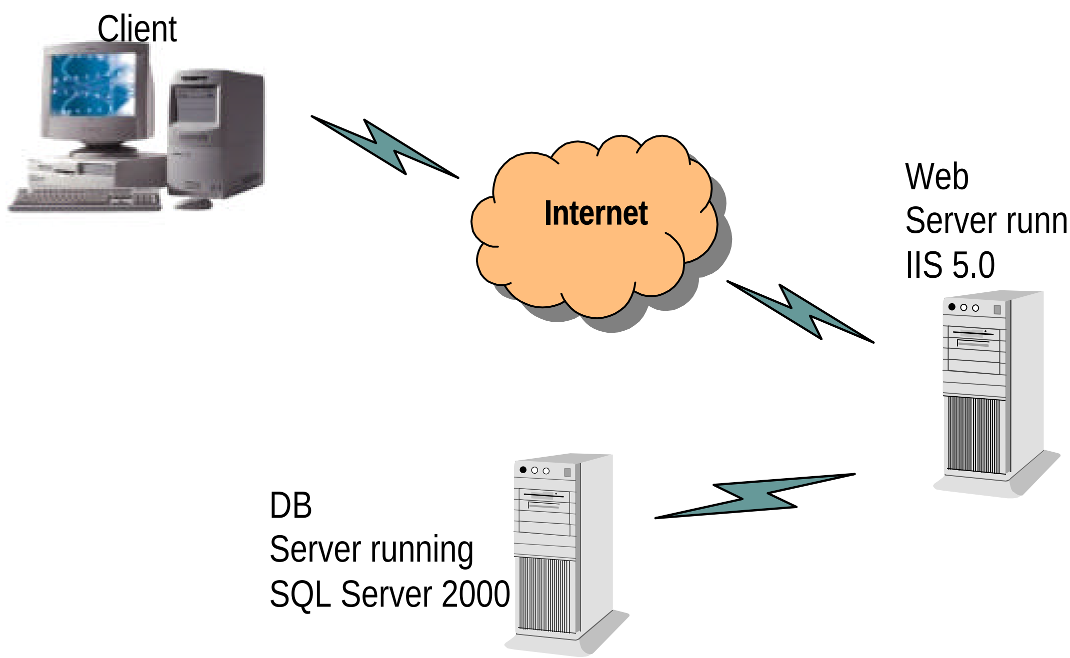


Figure 55. 3-Tier Architecture-Manpower Database.

3-tier architecture meets the requirements of large-scale Internet or intranet client/server applications because they are scalable, robust and flexible. They are easier to manage and deploy on the network, since most of the code runs on the servers.

3-tier applications minimize network interchanges by creating abstract levels of service. Instead of interacting with the database directly, the client calls business logic which resides on the server. The business logic then accesses the database on the client's behalf (middleware functionality).

For the thesis model specifically, almost all the logic of the architecture is concentrated on the database server side. This means that the network load is low since the only thing the web server does is to send commands to the database server on the client's behalf. These commands activate stored procedures on the database server's side, which do the entire job. Only the results of these procedures are sent to the client. The web server functions as a go-between between the client and the database server.

B. WEBSITE STRUCTURE

The website structure is based on the tasks that three types of users (officer, command and detailer) want to perform. The web design tool that is used for that purpose is the Macromedia Dreamweaver MX. The website administration is managed through the Microsoft IIS 5.0 server.

In order for the application to communicate with the database, an interface called Open Database Connectivity Driver (ODBC) must be installed first. ASP applications are fluent ODBC speakers thanks to a built-in OLE DB/ODBC interpreter.

The figure below shows the ODBC connectivity for the Manpower database. The name of the connection is 'LocalServer' since the SQL Server 2000 resides in the same computer.

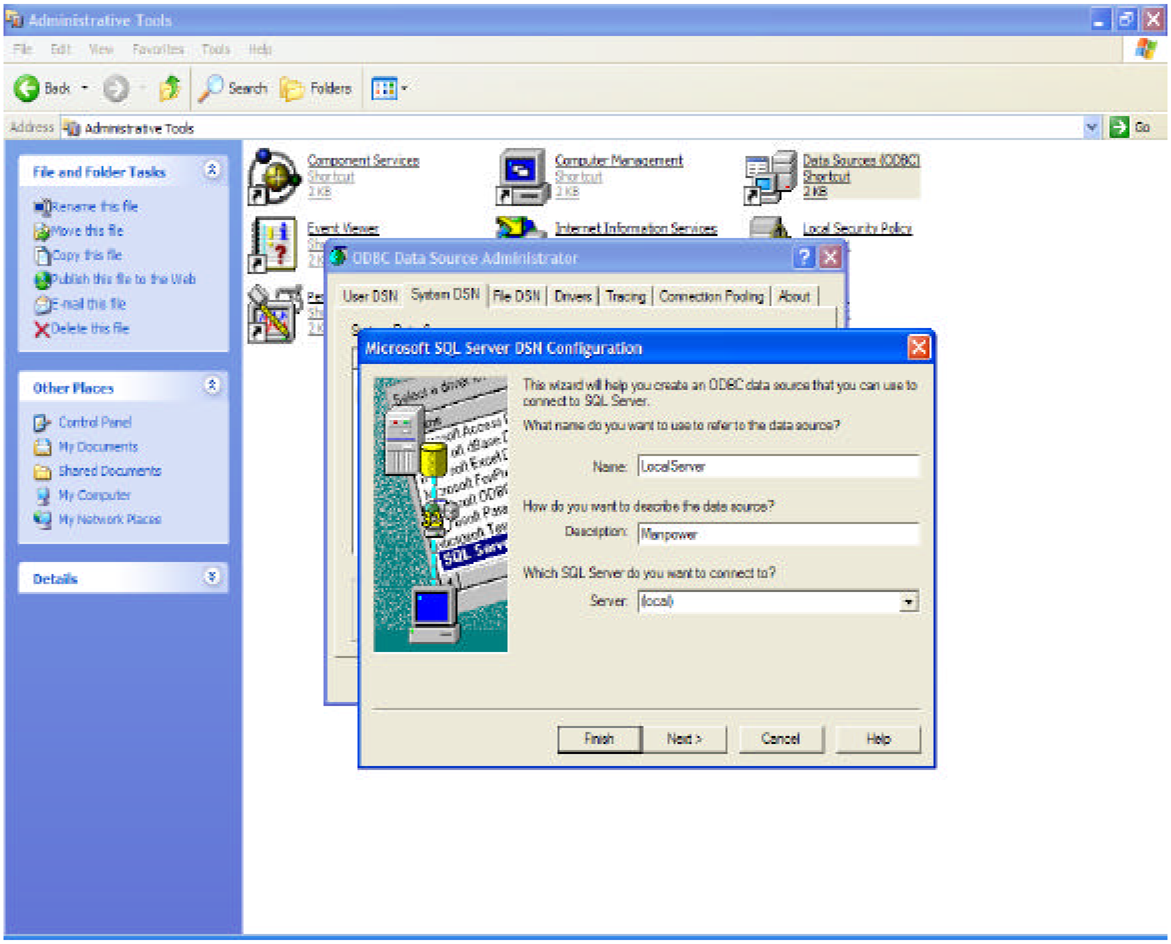


Figure 56. ODBC connectivity-Manpower Website.

The Manpower site is the place where all web pages are stored. The figure below shows the configurations of the Manpower website.

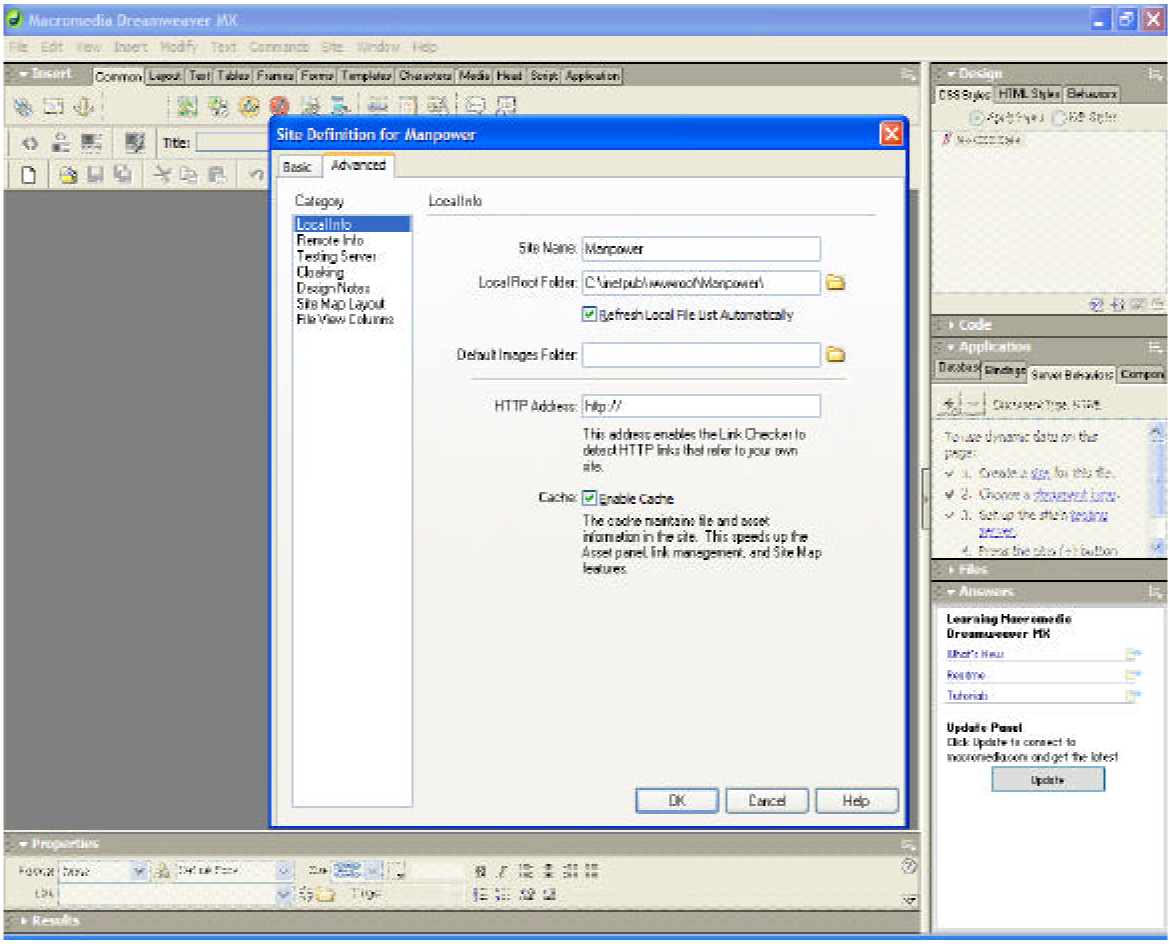


Figure 57. Manpower Website Configuration Wizard.

In order for the website to connect to the database a Data Source Name (DSN) should be created. A DSN is a one-word identifier that points to the database and contains all the information needed to connect to it. A DSN can be used if the connection is made through an ODBC driver. Below is the DSN for the Manpower website. This DSN string contains not only the ODBC connection named 'LocalServer', but also the user name and password of the administrator who creates the connection. After the connection is created successfully, then the web site administrator/creator has all the Manpower database components (tables, stored procedures etc.) available as shown at the right hand side of the figure.

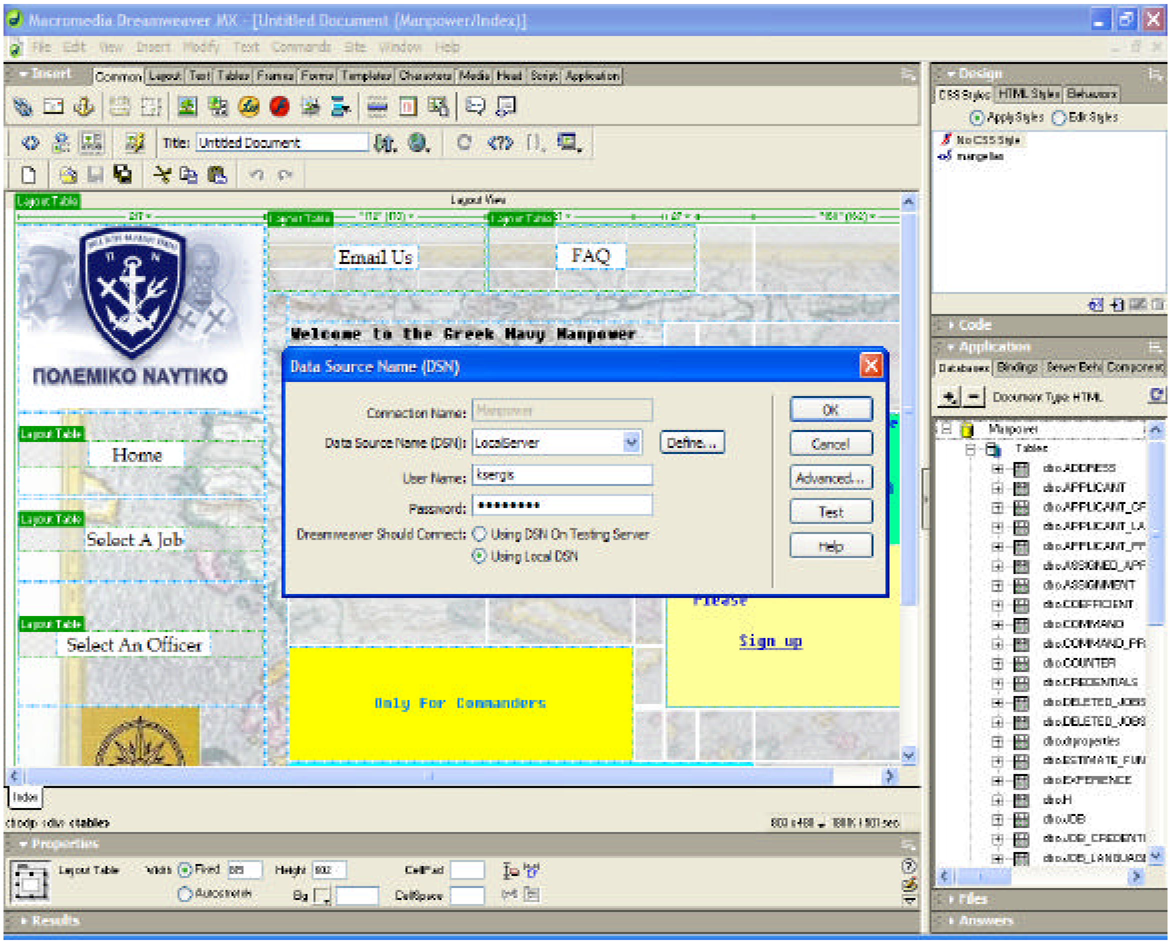


Figure 58. DSN Connection-Manpower Website.

Dreamweaver allows the administrator to create a recordset from which to extract dynamic content. A recordset is the result of a database query. It extracts the specific information the user requests and allows the user to display that information within a specified page.

Since almost all the functionality resides on the database server side, the administrator can use any stored procedures in order to define the kind of recordset the administrator wants for the webpage.

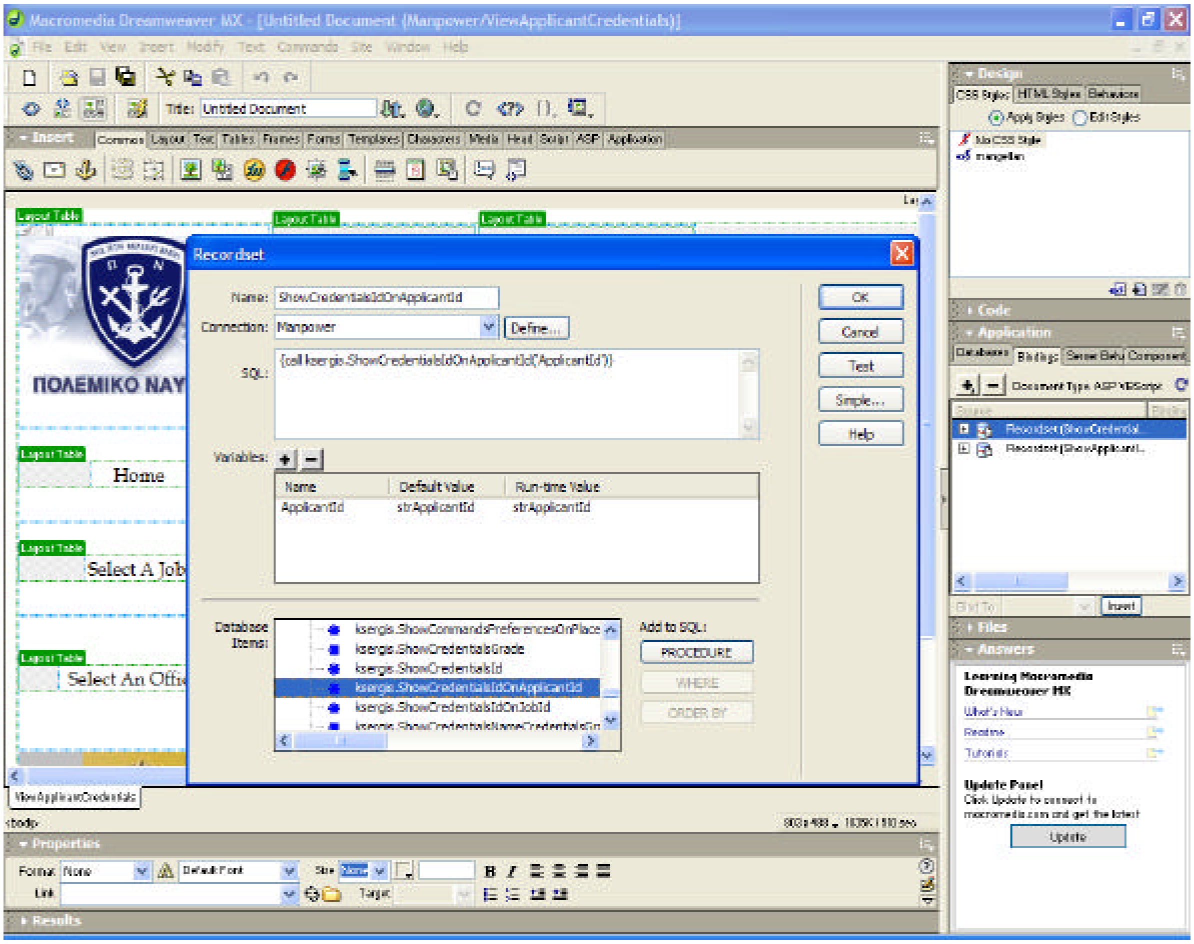


Figure 59. Recordset Based on the ksergis.ShowCredentialsIdOnApplicantId Stored Procedure-Manpower Website.

Dreamweaver allows the administrator to create interactive forms in order to allow the user to input his/her information in the webpage and store them in the database. For that purpose Dreamweaver has Form components collected in a bar (Form bar). The administrator can choose any component by performing a simple click. The most popular components are the following.

- Form inserts a form in the document. Dreamweaver inserts opening and closing form tags in the HTML source code. Any additional form objects, such as text fields, buttons, and so on must be inserted between the form tags for the data to be processed correctly by all browsers.
- Text Field inserts a text field in a form. Text fields accept any type of alphanumeric entries. The entered text can be displayed as a single line, as multiple lines, or as bullets or asterisks (for password protection).
- Field inserts a field in the document in which user data can be stored. Hidden fields let the administrator store information entered by a user,

such as a name, e-mail address, or purchase preference, and then use that data when the user next visits the site.

- Check Box inserts a check box in a form. Check boxes allow multiple responses in a single group of options. A user can select as many options as apply.
- Radio Button inserts a radio button in a form. Radio buttons represent exclusive choices. Selecting a button within a group deselects all others in the group. For example a user can select Yes or No.
- Radio Group inserts a collection of radio buttons which share the same name.
- List/Menus allows the administrator to create user choices in a list. The List option displays the option values in a scrolling list and allows users to select multiple options in the list. The Menu option displays the option values in a pop-up menu and allows users to select only a single choice.
- Button inserts a text button within a form. Buttons perform tasks when clicked, such as submitting or resetting forms. The administrator can add a custom name or label to a button, or use one of the predefined “Submit” or “Reset” labels.

The figure below shows a webpage of the Manpower website. This webpage contains a Form, a List/Menu, two Hiddenfields and two buttons (one called Update and one called Reset).

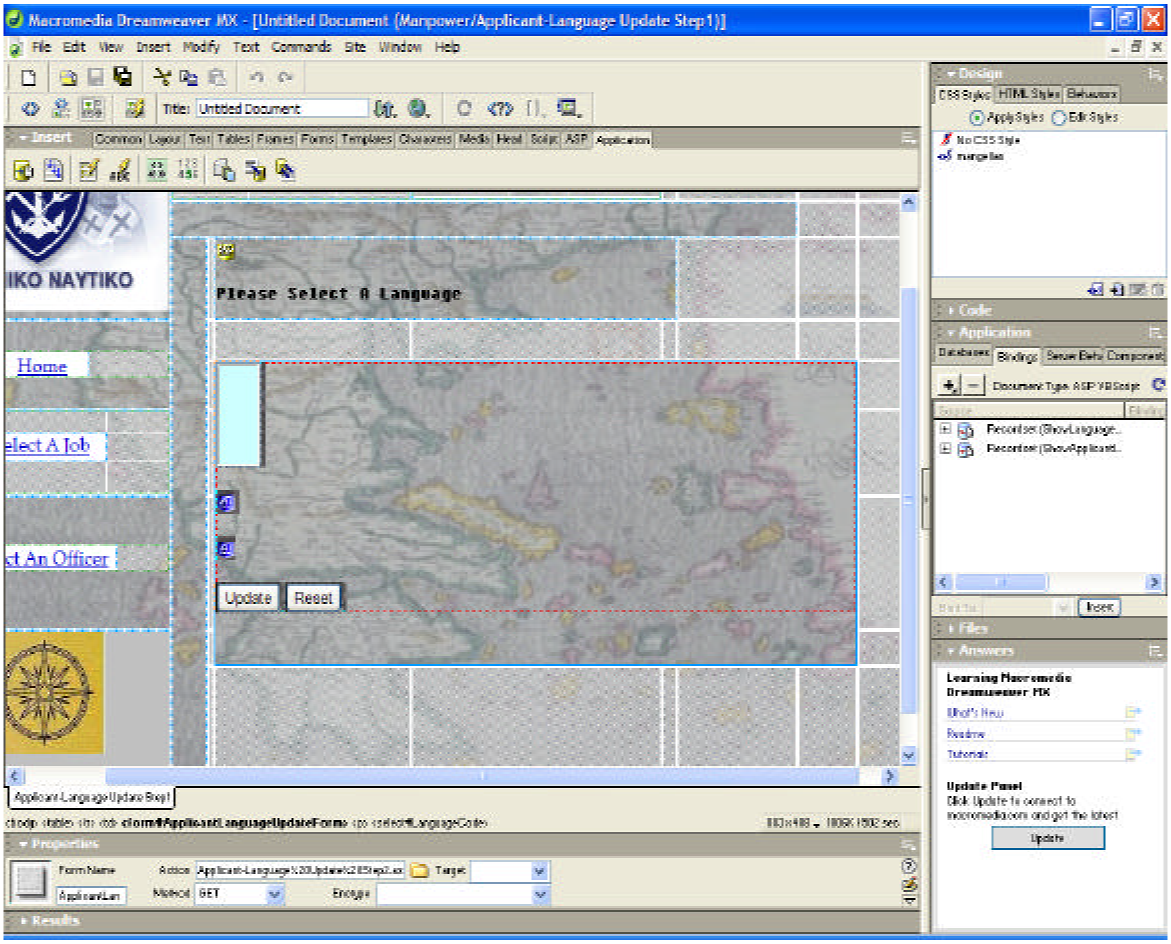


Figure 60. Webpage with a Form-Manpower Website.

One feature of Dreamweaver is the ability to build master pages. A master page is a page that lists records. For that purpose, Dreamweaver provides the webpage designer with a special bar named 'Application'. The most popular components of the 'Application' bar are the following.

- Repeated Region displays more than one record at a time. The repeated region is normally applied to the table row containing the dynamic content.
- Dynamic Table creates the table row and the repeated region automatically.
- Recordset Navigation Bar helps the user to navigate through all the records.

The following figures display the 'ViewCredentialInfo.asp' page in both the Dreamweaver and Internet environment.

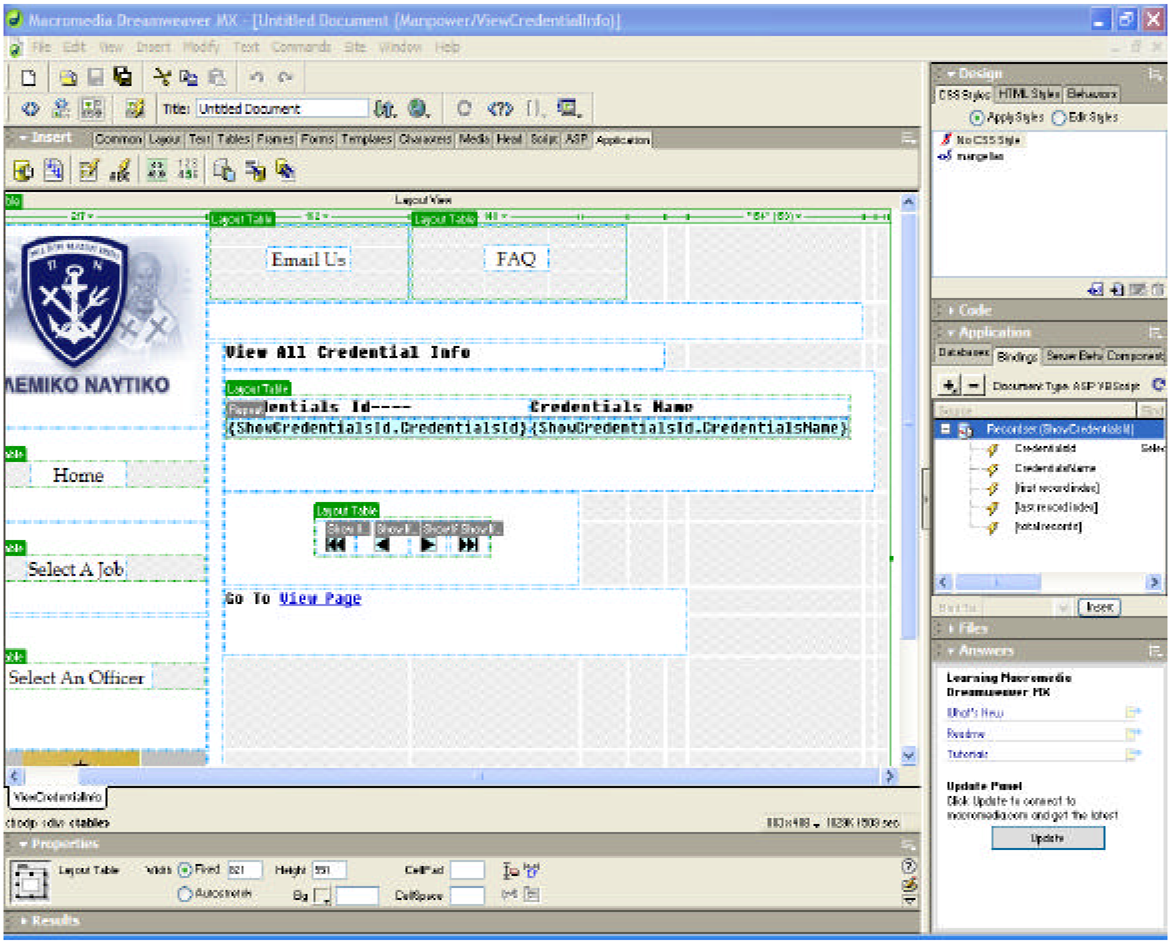


Figure 61. Master Page-The Repeated Region and the Navigation Bar Are Displayed.

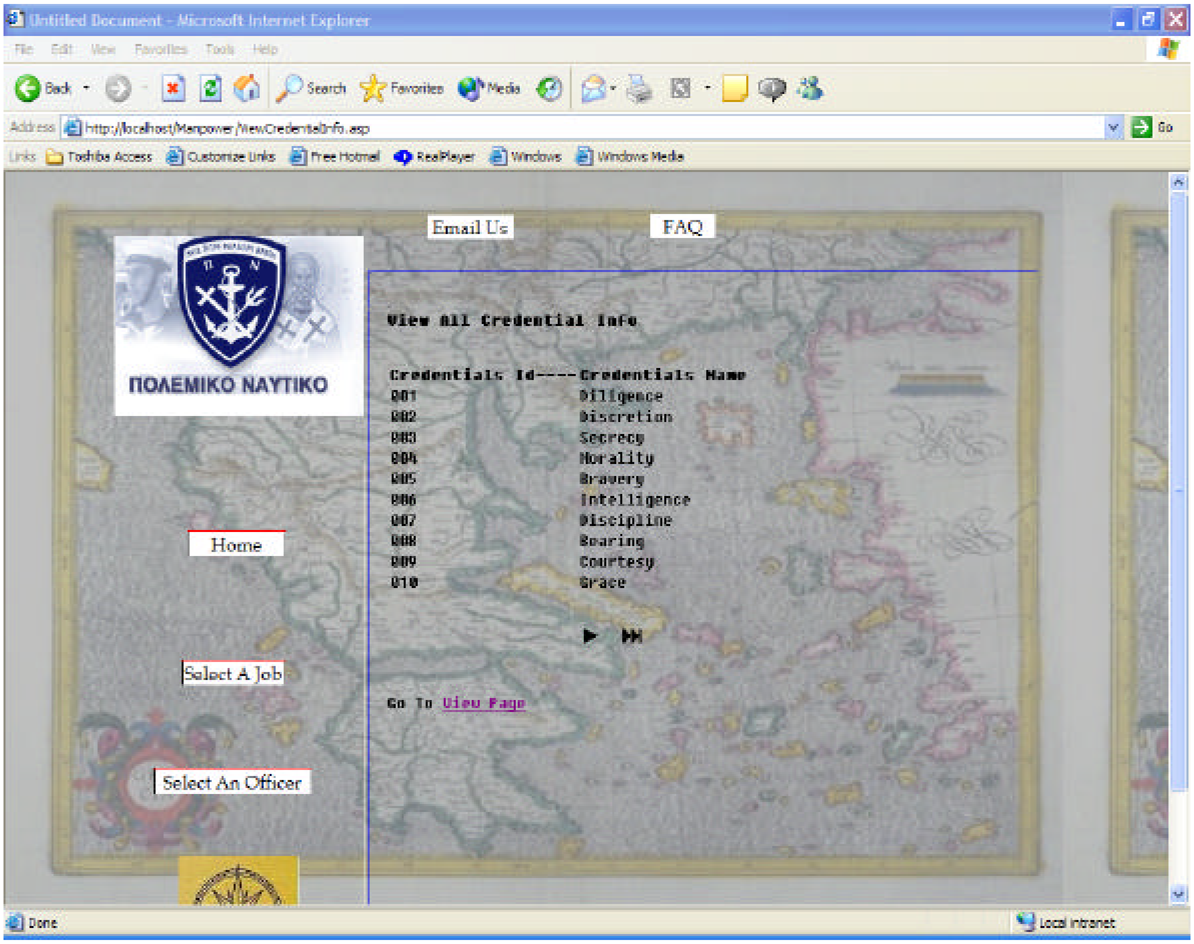


Figure 62. Master Page (1st Screen)-How the Repeated Region and the Navigation Bar Are Displayed on the Internet.

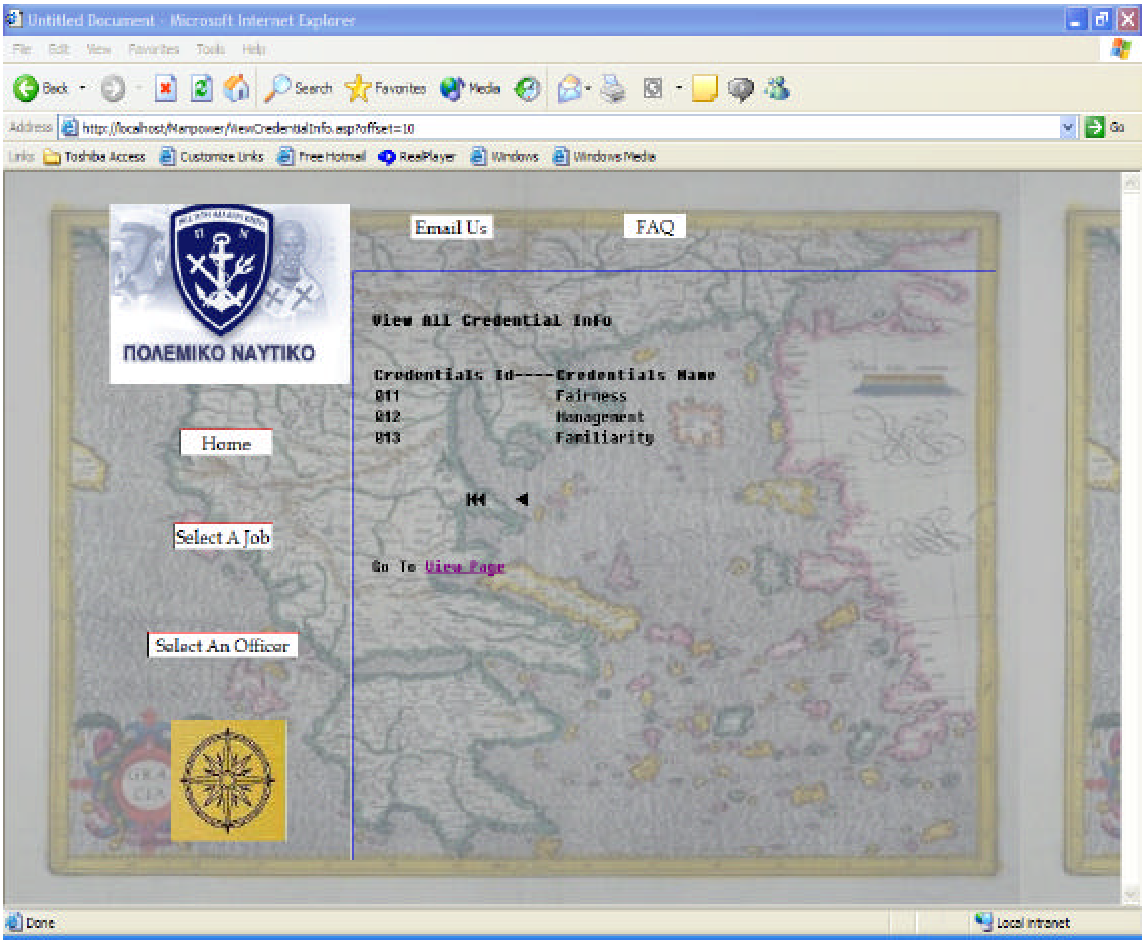


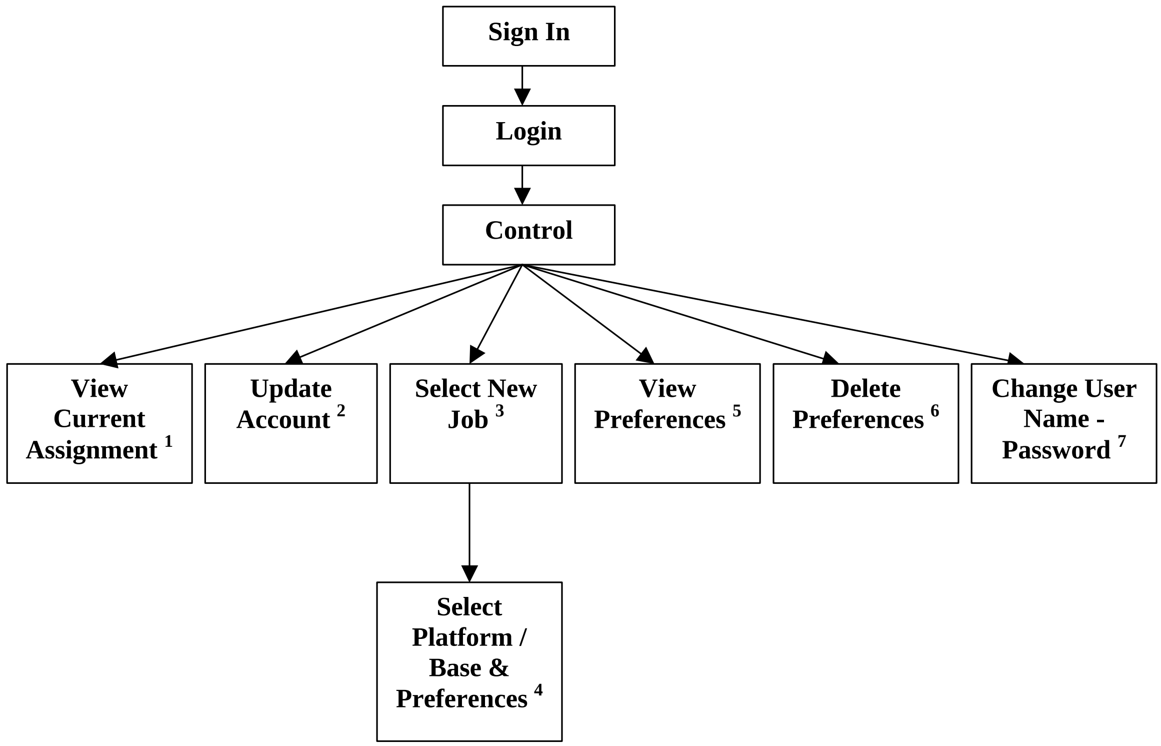
Figure 63. Master Page (2nd Screen)-How the Repeated Region and the Navigation Bar are Displayed on the Internet.

C. MENU NAVIGATIONAL TREE

The three categories of users determine the shape and structure of the Manpower website. These categories of users are the officer, the command and the detailer. The officer has to declare his preferences for the next assignment. The command has to declare its preferences for the officers who wants to occupy one of their jobs. The detailer has control of the website. The detailer has to view all the records of the Manpower database, update them or delete them. The detailer also has to solve the assignment problem and change the solution according to the Navy's desires.

The following lines present a description of the sequence of actions each one of the users has to perform in order to accomplish his/her role in the Manpower website. Each step has a corresponding number of stored procedures that are executed. These are also presented in this section.

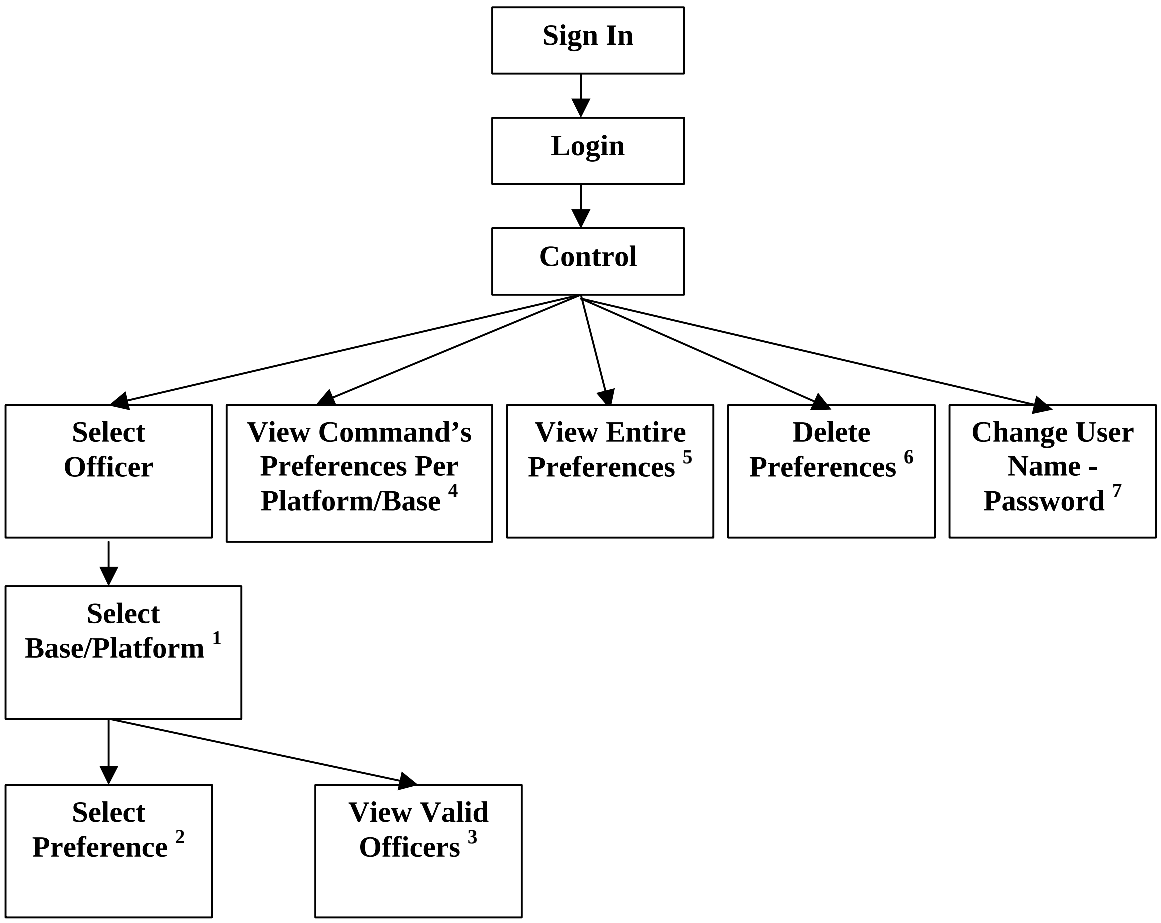
1. Officer



Stored Procedures for Officer			
#	**Name**	**Variables**	**Description**
1	ShowCurrentAssignment	ApplicantId	Returns the officer's current assignment
2	ShowApplicantAddressPhoneData	ApplicantId	Returns the officer's address and phone information
	UpdateApplicantData	ApplicantId, FirstName, LastName, MiddleName, EmailAddress	Updates the officer's First Name, Last Name, Middle Name, Email Address
	UpdateAddressData	ApplicantId, CityOrTown, Street, Apartment, ZIP	Updates the City or Town, Street, Apartment and ZIP code the officer lives in
	UpdatePhoneData	ApplicantId, HomePhoneNumber, CellPhoneNumber, OtherPhoneNumber	Updates the officer's Home Phone Number, Cell Phone Number and Other Phone Number
3	ShowJobId		Returns all the jobs
	CheckApplicantSuitable	ApplicantId	Returns the jobs the officer is suitable for
4	ShowPlaceCodeOnJobId	JobId	Returns the Platform/Base data per job
	CheckPreference	ApplicantId, PreferenceApplicant, PlaceCode, JobId	Checks if the officer has selected the same Preference number or Platform/Base
5	CheckApplicantPreferenceExists	ApplicantId	Checks if the officer has at least one Preference
	ShowApplicantPreferences	ApplicantId	Returns all the officer's Preferences

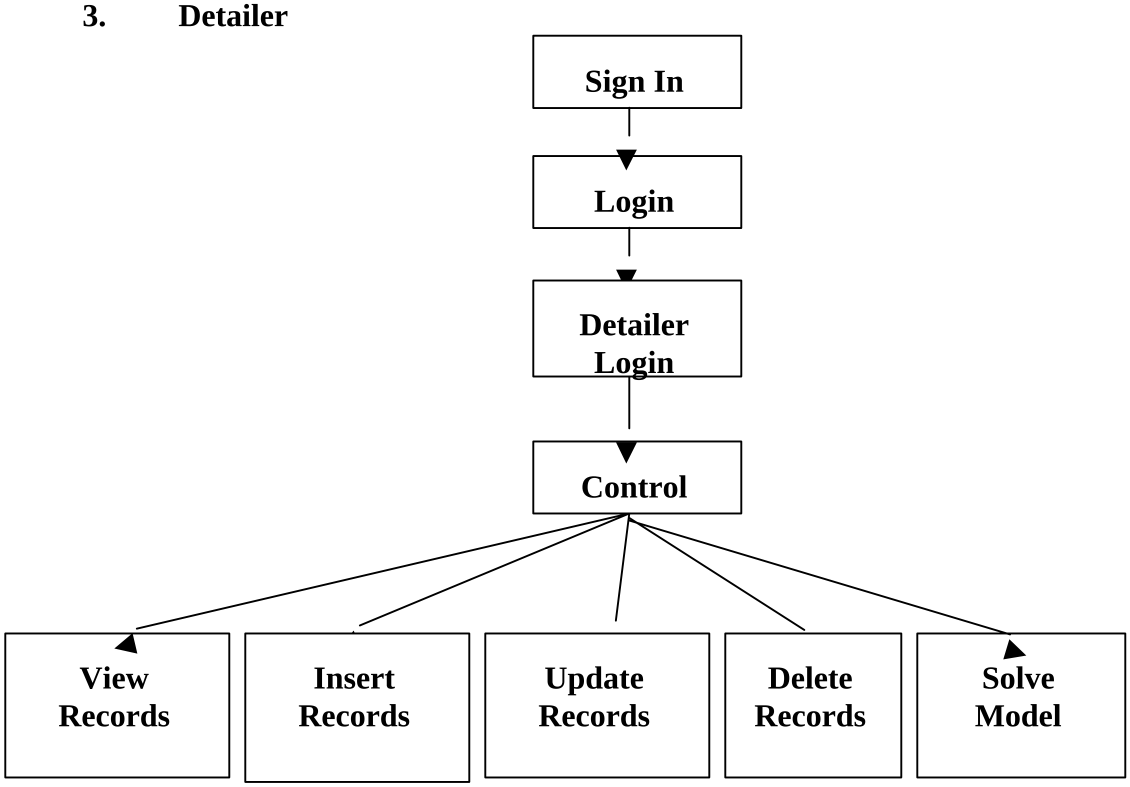
Stored Procedures for Officer			
#	**Name**	**Variables**	**Description**
6	ShowApplicantPreferences	ApplicantId	Returns all the officer’s Preferences
	DeleteApplicantPreference	ApplicantId, PreferenceApplicant	Deletes an officer’s Preference
7	CheckUserName	UserName	Checks if the User Name is unique
	UpdateUserNamePassword	ApplicantId, UserName, Password	Updates the officer’s User Name and Password

2. Command

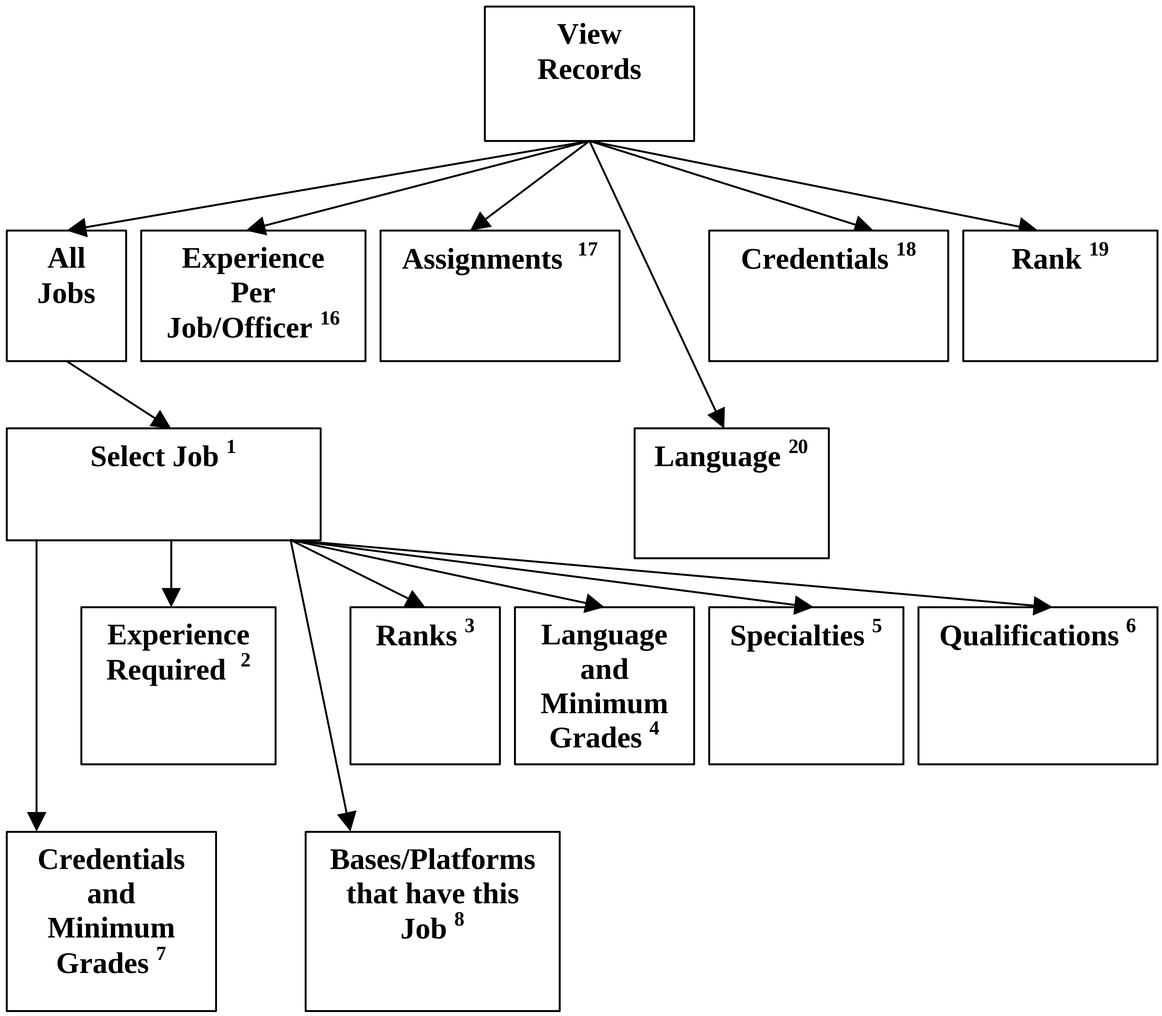


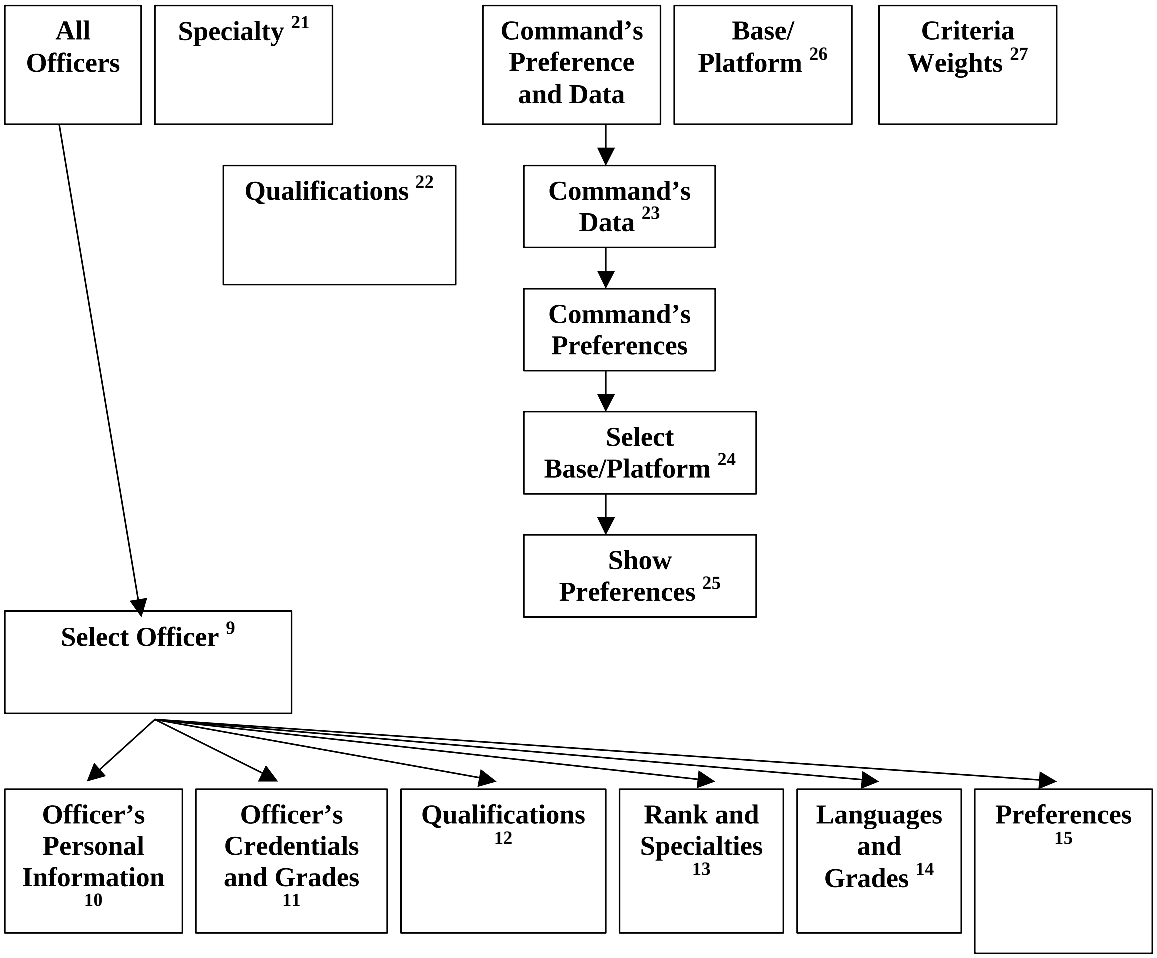
Stored Procedures for Command			
#	Name	Variables	Description
1	ShowPlaceCodeOnCommandCode	CommandCode	Returns the Platform/Base code per command
2	ShowJobIdOnPlaceCode	PlaceCode	Returns the jobs per Platform/Base
	ShowApplicantLastNameFirstName		Returns the Officer's First Name and Last Name
	CheckPreferenceCommand	CommandCode, ApplicantId, PreferenceCommand, PlaceCode, JobId	Checks if the command has selected the same Preference number or the same officer and Platform/Base twice
3	ShowJobIdOnPlaceCode	PlaceCode	Returns the jobs per Platform/Base
	CheckSuitableApplicantsOnJob	JobId	Returns the officers that are eligible for a job
4	ShowPlaceImage	CommandCode	Returns the Platform/Base jpeg files per command
	ShowCommandPreferencesOnPlaceCode	CommandCode, PlaceCode	Returns the command's preferences per Platform/Base
5	ShowCommandPreferences	CommandCode	Returns the command's preferences
6	ShowCommandsPreferencesForDelete	CommandCode	Returns the command's preferences
	DeleteCommandPreference	PlaceCode, JobId, PreferenceCommand, ApplicantId	Deletes a command's preference
7	CheckUserNameCommand	UserName	Checks if the User Name is unique
	UpdateUserNamePasswordCommand	CommandCode, UserName, Password	Updates the command's User Name and Password

3. Detailer



a. View Records

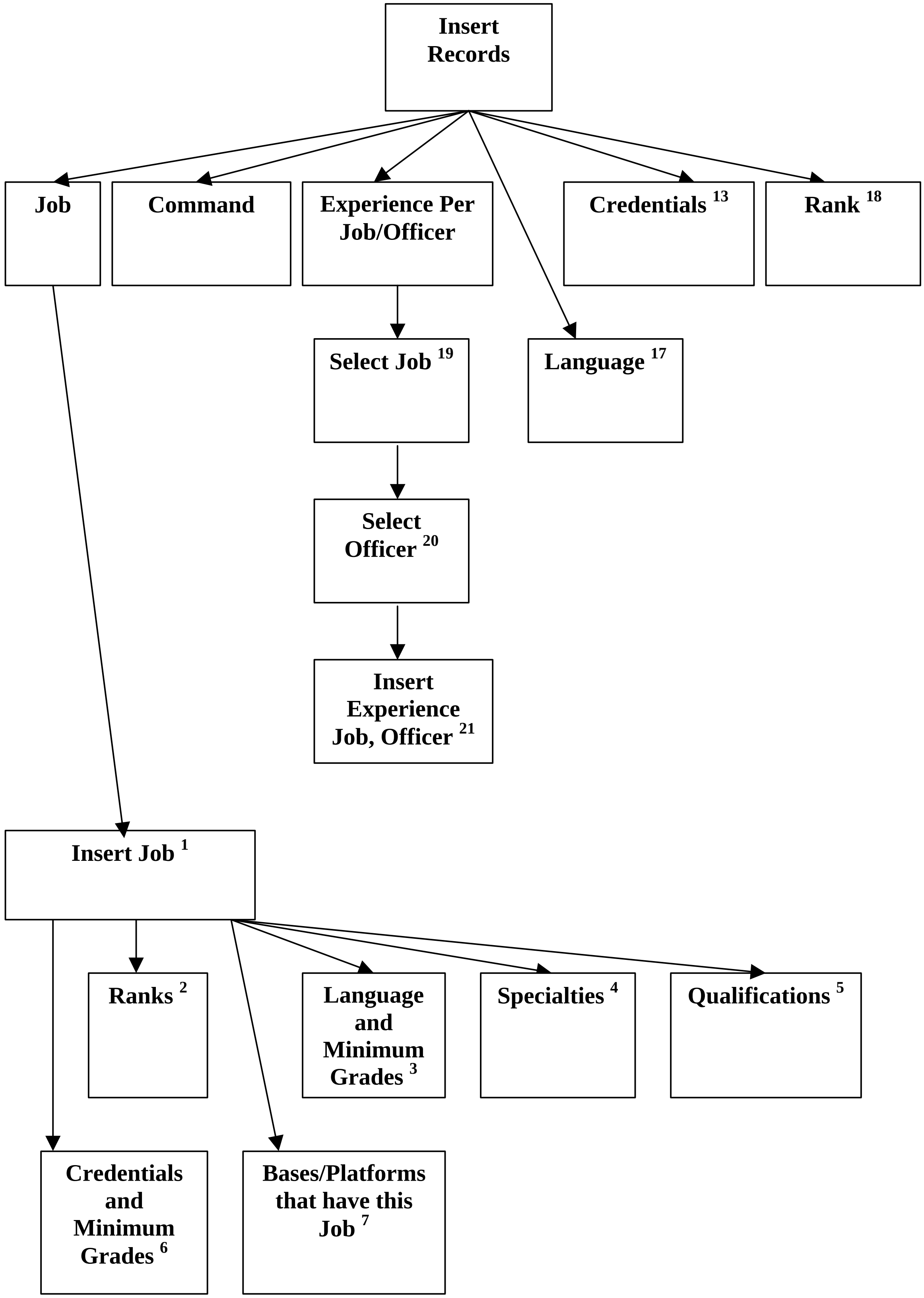


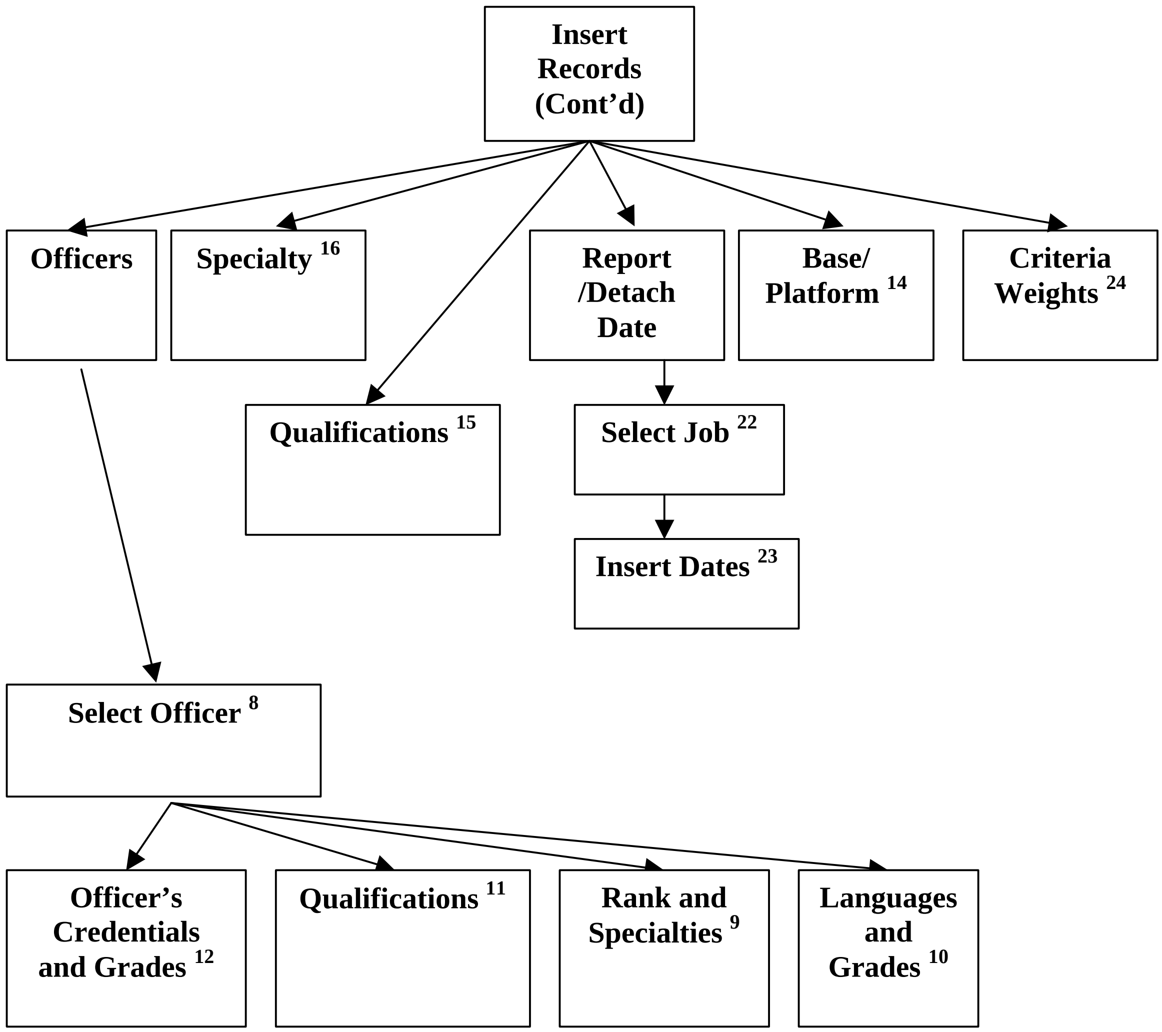


Stored Procedures for View Records			
#	**Name**	**Variables**	**Description**
1	ShowJobId		Returns all jobs
2	ShowExperienceRequired	JobId, JobName	Returns the required experience per job
3	ShowRankNameTimeSeaServiceOnJobId	JobId	Returns the rank and time of sea service per job
4	ShowLanguageNameLanguageDegreeOnJobId	JobId	Returns the language and its minimum grades per job
5	ShowSpecialtyNameOnJobId	JobId	Returns the name of the specialty per job
6	ShowQualificationNameOnJobId	JobId	Return the qualification's name per job
7	ShowCredentialsNameCredentialsGradeOnJobId	JobId	Returns the credential and its minimum grades per job
8	ShowPlaceNamePlaceImageCommandNameOnJobId	JobId	Returns the platforms' name, jpeg file and command per job
9	ShowApplicantIdLastNameFirstNameWORank		Returns the officer's last name and first name
10	ShowApplicantAddressPhoneData	ApplicantId	Returns the officer's address and phone data

Stored Procedures for View Records			
#	**Name**	**Variables**	**Description**
11	ShowCredentialsIdOnApplicantId	ApplicantId	Returns the credentials and the corresponding grades per officer
	ShowApplicantIdLastNameFirstNameOnApplicantId	ApplicantId	Returns the officer's first and last name
12	ShowApplicantIdLastNameFirstNameOnApplicantId	ApplicantId	Returns the officer's first and last name
	ShowQualificationCodeOnApplicantId	ApplicantId	Returns the qualifications per officer
13	ShowApplicantIdLastNameFirstNameOnApplicantId	ApplicantId	Returns the officer's first and last name
	ShowRankCodeSpecialtyCodeSeaServiceOnApplicantId	ApplicantId	Returns the officer's rank, specialty and sea service
14	ShowApplicantIdLastNameFirstNameOnApplicantId	ApplicantId	Returns the officer's first and last name
	ShowLanguageCodeOnApplicantId	ApplicantId	Returns the officer's languages and the corresponding grades
15	ShowApplicantIdLastNameFirstNameOnApplicantId	ApplicantId	Returns the officer's first and last name
	ShowApplicantPreferences	ApplicantId	Returns the officer's preferences
16	ShowExperiencePerJobOfficer		Returns the officer's experience per job
17	ShowAllAssignmentInfo		Returns all the assignments
18	ShowCredentialsId		Returns all the credentials
19	ShowRankData		Returns all the ranks
20	ShowLanguageCode		Returns all the languages
21	ShowSpecialtyCode		Returns all the specialties
22	ShowQualificationCode		Returns all the qualifications
23	ShowCommandsData		Returns all the commands
24	ShowPlaceImage	CommandCode	Returns the jpeg files of all the platforms /bases per command
25	ShowCommandsPreferencesOnPlaceCode	CommandCode, PlaceCode	Returns the command's preferences per platform /base
26	ShowPlaceData		Returns all the platforms /bases
27	ShowCoefficients		Returns all the coefficients with their weights

b. Insert Records





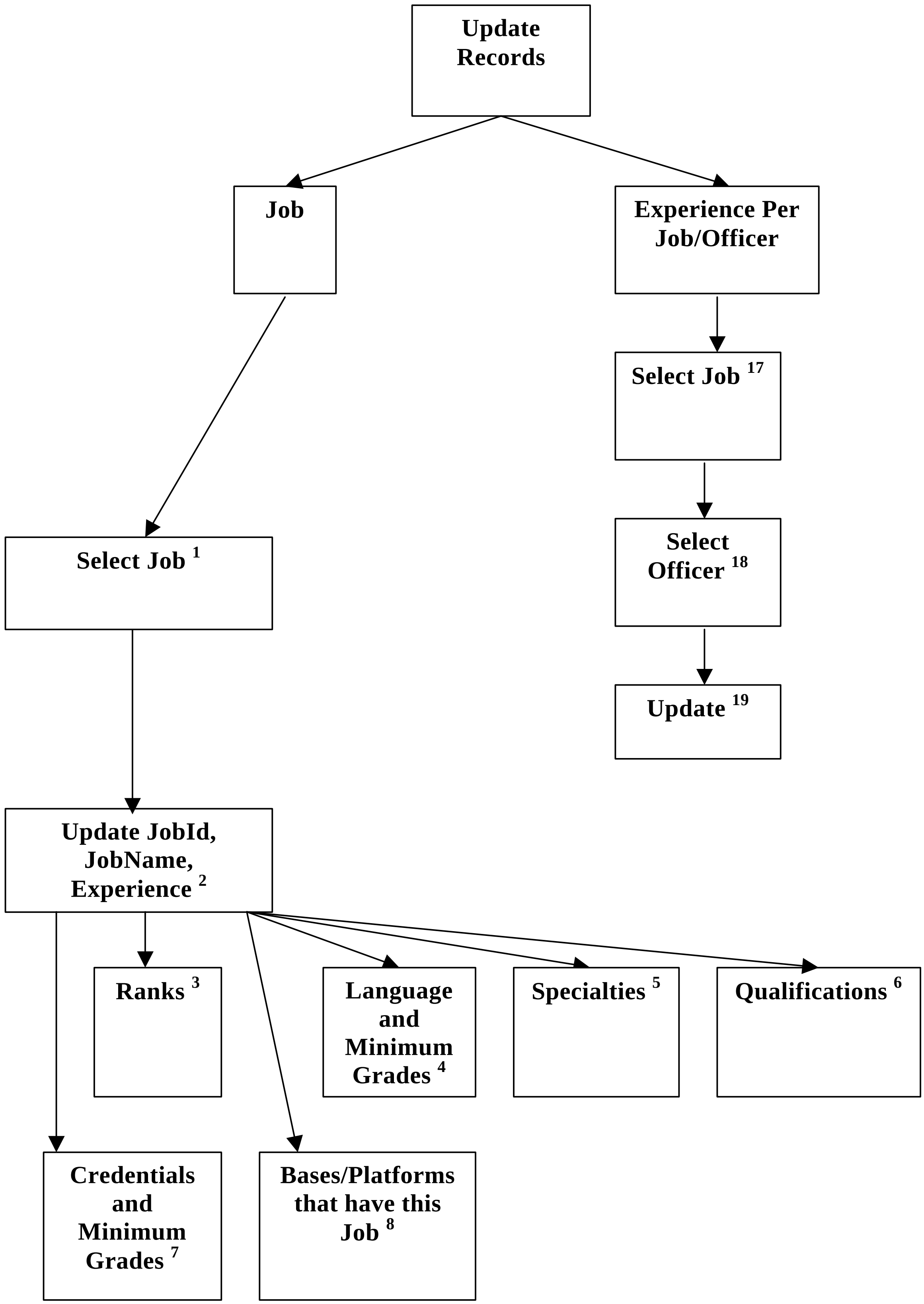
Stored Procedures for Insert Records

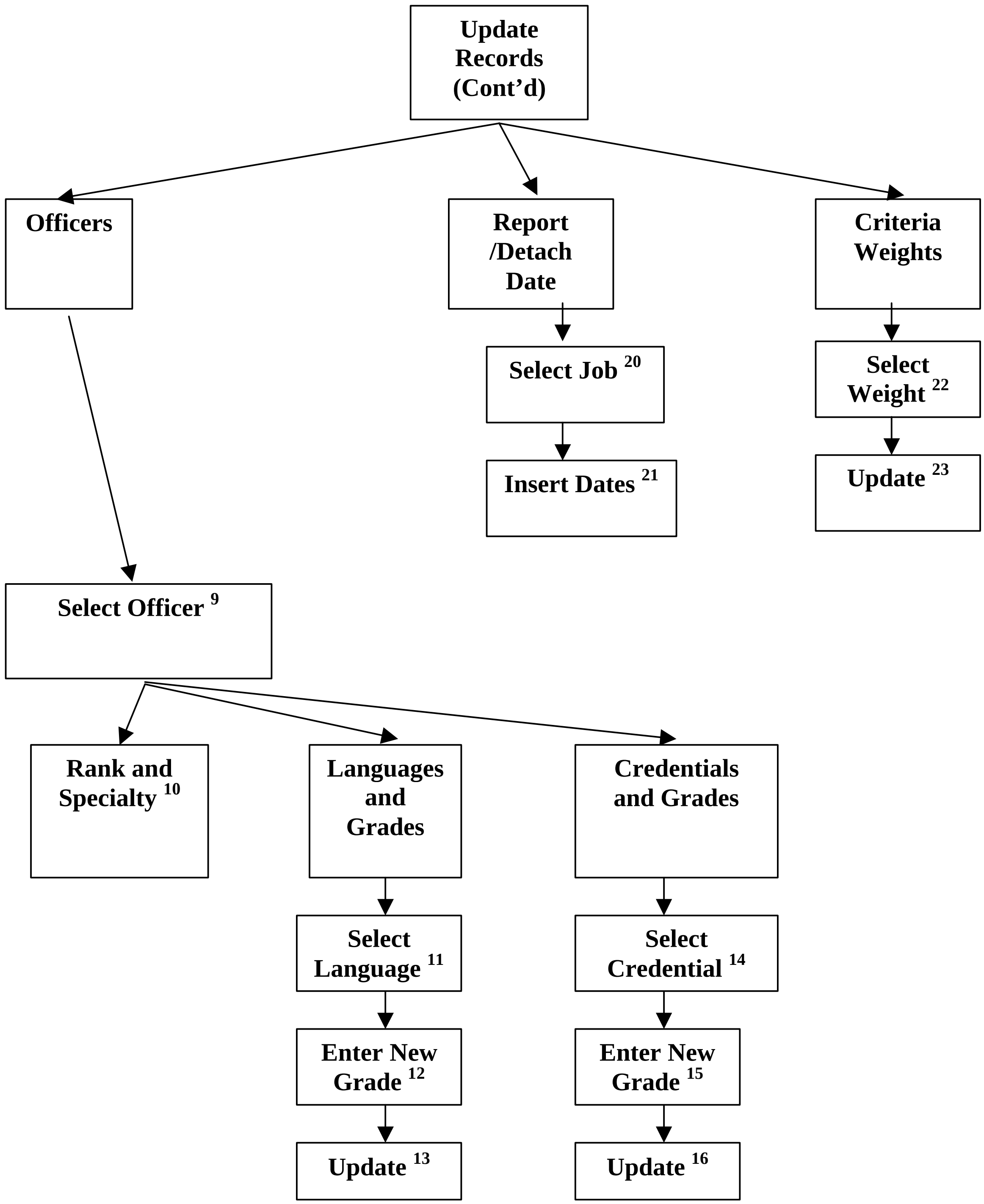
#	Name	Variables	Description
1	ShowJobId		Returns all jobs
	CheckJobId	JobId	Checks if the JobId is unique
	CheckJobName	JobName	Checks if the JobName is unique
2	ShowRankCode		Returns all the ranks
	CheckJobIdRankCode	JobId, RankCode	Checks if the JobId, RankCode pair exists
3	ShowLanguageCode		Returns all the languages
	CheckJobIdLanguageCode	JobId, LanguageCode	Checks if the JobId, LanguageCode pair exists
4	ShowSpecialtyCode		Returns all the specialties
	CheckJobIdSpecialtyCode	JobId, SpecialtyCode	Checks if the JobId, SpecialtyCode pair exists
5	ShowQualificationCode		Returns all the qualifications
	CheckJobIdQualificationCode	JobId, QualificationCode	Checks if the JobId, QualificationCode pair exists
6	ShowCredentialsId		Returns all the credentials

Stored Procedures for Insert Records			
#	**Name**	**Variables**	**Description**
	CheckJobIdCredentialsId	JobId, CredentialsId	Checks if the JobId, CredentialsId pair exists
7	ShowPlaceCode		Returns all the Platforms/Bases
	CheckJobIdPlaceCode	JobId, PlaceCode	Checks if the JobId, PlaceCode pair exists
8	ShowApplicantIdLastNameFirstNameWORank		Returns the officer's last name and first name
9	ShowRankCode		Returns all the ranks
	ShowSpecialtyCode		Returns all the specialties
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the last name, first name and rank per officer
	UpdateApplicantIdSpecialtyRank	ApplicantId, SpecialtyCode, RankCode, SeaTimeForRank	Updates the specialty, rank and required sea time for the rank per officer
10	ShowLanguageCode		Returns all the languages
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the last name, first name and rank per officer
	CheckApplicantIdLanguageCode	ApplicantId, LanguageCode	Checks if the ApplicantId, LanguageCode pair exists
11	ShowQualificationCode		Returns all the qualifications
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the last name, first name and rank per officer
	CheckApplicantIdQualificationCode	ApplicantId, QualificationCode	Checks if the ApplicantId, QualificationCode pair exists
12	ShowCredentialsId		Returns all the credentials
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the last name, first name and rank per officer
	CheckApplicantIdCredentialsId	ApplicantId, CredentialsId	Checks if the ApplicantId, CredentialsId pair exists
13	CheckCredentialsId	CredentialsId	Checks if the CredentialsId is unique
	CheckCredentialsName	CredentialsName	Checks if the CredentialsName is unique
14	ShowCommandCode		Returns all Command Codes
15	CheckQualificationCode	QualificationCode	Checks if the QualificationCode is unique
	CheckQualificationName	QualificationName	Checks if the QualificationName is unique
16	CheckSpecialtyCode	SpecialtyCode	Checks if the SpecialtyCode is unique
	CheckSpecialtyName	SpecialtyName	Checks if the SpecialtyName is unique
17	CheckLanguageCode	LanguageCode	Checks if the LanguageCode is unique
	CheckLanguageName	LanguageName	Checks if the LanguageName is unique
18	CheckRankCode	RankCode	Checks if the RankCode is unique
	CheckRankName	RankName	Checks if the RankName is unique
19	ShowJobId		Returns all jobs
20	CheckSuitableApplicantsOnJobId	JobId	Checks if an officer is eligible for a job
21	CheckExperienceExists	JobId, ApplicantId	Checks if an experience has been already inserted
	InsertExperience	JobId, ApplicantId, Experience	Inserts the experience the officer has for a job
22	ShowJobIdPlaceCodeApplican		Returns all the assignments

Stored Procedures for Insert Records			
#	**Name**	**Variables**	**Description**
	tIdFromASSIGNMENT		
23	ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENT	ApplicantId	Returns an officer’s assignment
	CheckDateExists	ApplicantId	Checks if the report or detach date exists
	InsertDate	ApplicantId, ReportDate, DetachDate	Inserts the Report and Detach Dates
24	CheckCoefficientExists	WeightName	Checks if the coefficient exists
	InsertCoefficient	WeightName, WeightValue	Inserts the coefficient and its value

c. Update Records

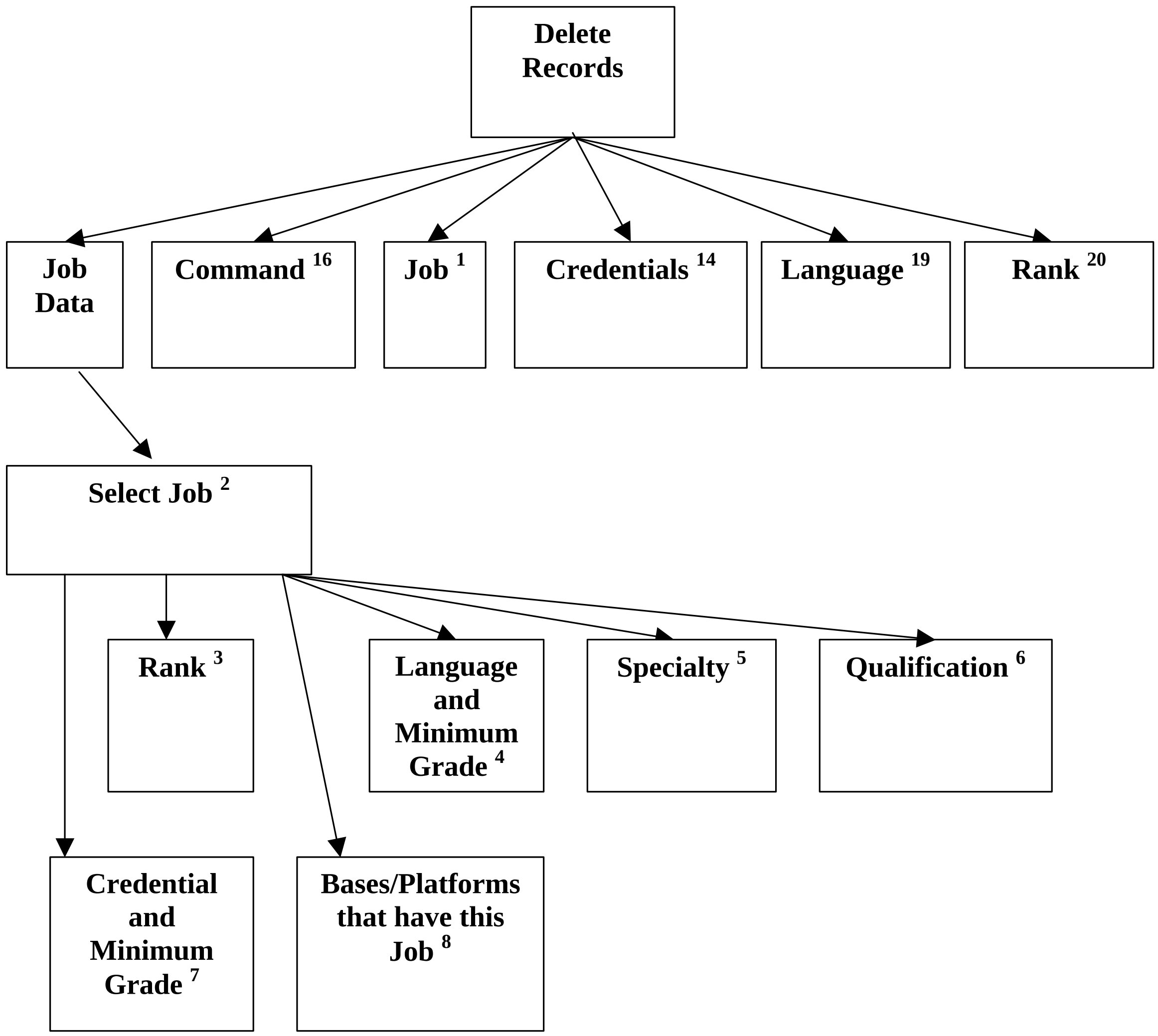


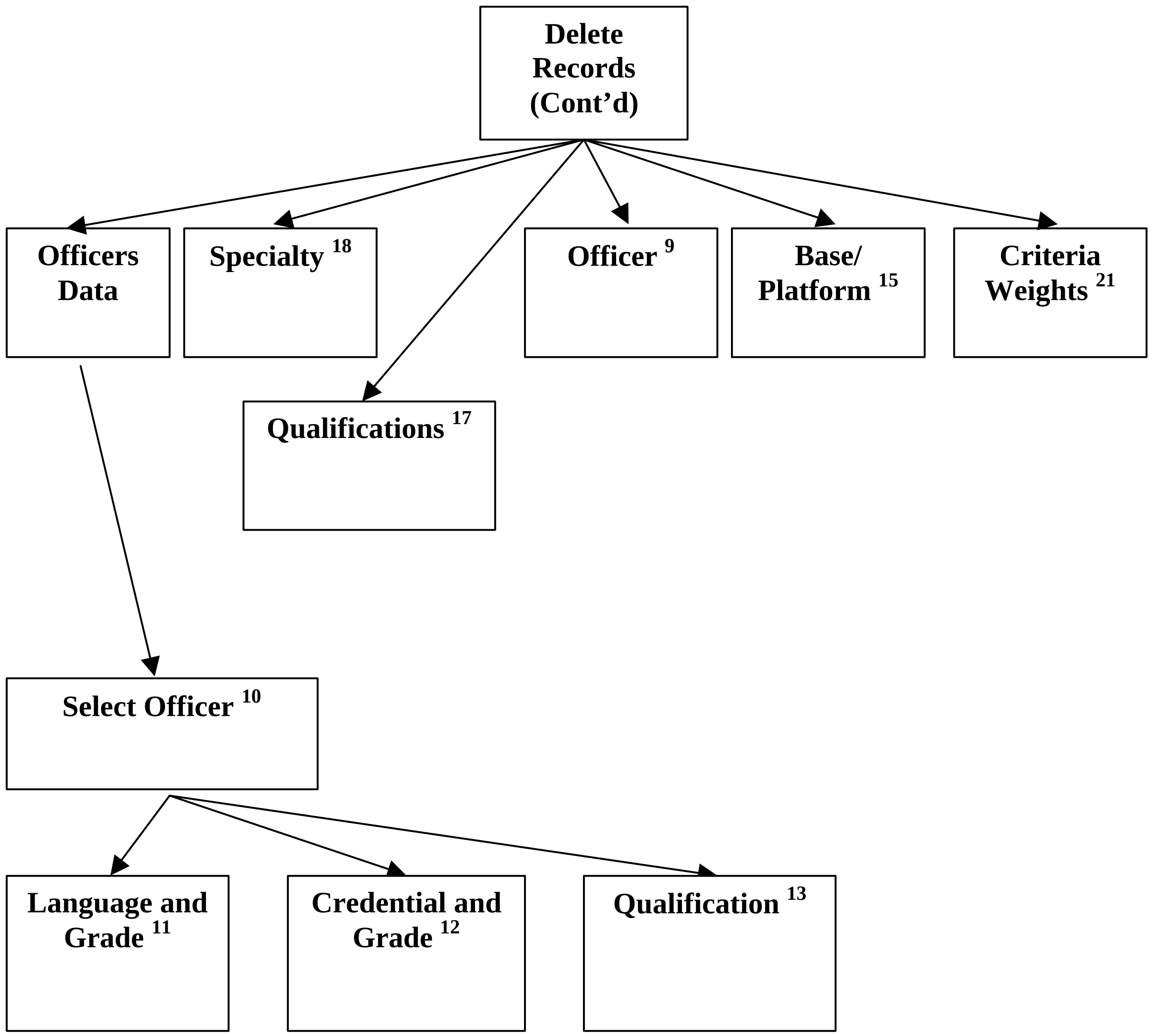

Update Records (Cont'd)
Officers
Report /Detach Date
Criteria Weights
Select Job [20]
Insert Dates [21]
Select Weight [22]
Update [23]
Select Officer [9]
Rank and Specialty [10]
Languages and Grades
Credentials and Grades
Select Language [11]
Enter New Grade [12]
Update [13]
Select Credential [14]
Enter New Grade [15]
Update [16]

Stored Procedures for Update Records			
#	Name	Variables	Description
1	ShowJobId		Returns all jobs
2	ShowJobId		Returns all jobs
	ShowExperienceRequired	JobId, JobName	Returns the job's experience required
	CheckJobId	JobIdNew	Checks if the new JobId is unique
	CheckJobName	JobNameNew	Checks if the new JobName is unique
	UpdateJobIdJobNameExperienceRequired	JobId, JobIdNew, JobNameNew, ExperienceRequired	Updates the JobId, the JobName and the experience required
3	ShowRankCode		Returns all the ranks
	CheckJobIdRankCode	JobId, RankCode	Checks if the JobId, RankCode pair exists
4	ShowLanguageCode		Returns all the languages
	CheckJobIdLanguageCode	JobId, LanguageCode	Checks if the JobId, LanguageCode pair exists
5	ShowSpecialtyCode		Returns all the specialties
	CheckJobIdSpecialtyCode	JobId, SpecialtyCode	Checks if the JobId, SpecialtyCode pair exists
6	ShowQualificationCode		Returns all the qualifications
	CheckJobIdQualificationCode	JobId, QualificationCode	Checks if the JobId, QualificationCode pair exists
7	ShowCredentialsId		Returns all the credentials
	CheckJobIdCredentialsId	JobId, CredentialsId	Checks if the JobId, CredentialsId pair exists
8	ShowPlaceCode		Returns all the Platforms/Bases
	CheckJobIdPlaceCode	JobId, PlaceCode	Checks if the JobId, PlaceCode pair exists
9	ShowApplicantIdLastNameFirstNameWORank		Returns the officer's last name and first name
10	ShowRankCode		Returns all ranks
	ShowSpecialtyCode		Returns all specialties
	ShowApplicantRankSpecialtySeaTimeForRank	ApplicantId	Returns the rank, specialty and sea time for rank per officer
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the officer's last name, first name and rank
	UpdateApplicantIdSpecialtyRank	ApplicantId, RankCode, SpecialtyCode, SeaTimeForRank	Updates the officer's rank, specialty, sea time for his/her rank
11	ShowLanguageCodeOnApplicantId	ApplicantId	Returns the officer's languages and grades
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the officer's last name, first name and rank
12	ShowLanguageDegree	ApplicantId, LanguageCode	Returns the officer's language and grade
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the officer's last name, first name and rank
13	UpdateLanguageDegree	ApplicantId, LanguageCode, LanguageDegree	Updates the officer's language grades

Stored Procedures for Update Records			
#	**Name**	**Variables**	**Description**
14	ShowCredentialsIdOnApplicantId	ApplicantId	Returns the credential grades per officer
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the officer’s last name, first name and rank
15	ShowCredentialsGrade	ApplicantId, CredentialsId	Returns the officer’s credential grade
	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns the officer’s last name, first name and rank
16	UpdateCredentialsGrade	ApplicantId, CredentialsId, CredentialsGrade	Updates the officer’s credential grade
17	ShowJobIdJobNameFromEXPERIENCE		Returns all jobs with their required experience
18	ShowApplicantDataOnJobFromEXPERIENCE	JobId	Returns the officers for a specific job
19	ShowExperienceOnJobIdJobName	JobId, ApplicantId	Returns the officer’s experience for a specific job
	UpdateExperience	JobId, ApplicantId, Experience	Updates the experience per job, officer
20	ShowJobIdPlaceCodeApplicantIdFromASSIGNMENTForUpdate		Returns all the assignments
21	ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENTForUpdate		Returns an officer’s assignment
	InsertDate	ApplicantId, ReportDate, DetachDate	Inserts the report and detach date for a specific officer
22	ShowCoefficients		Returns all the coefficients and their values
	UpdateCoefficient	WeightName, WeightValue	Updates the coefficients’ values

d. Delete Records

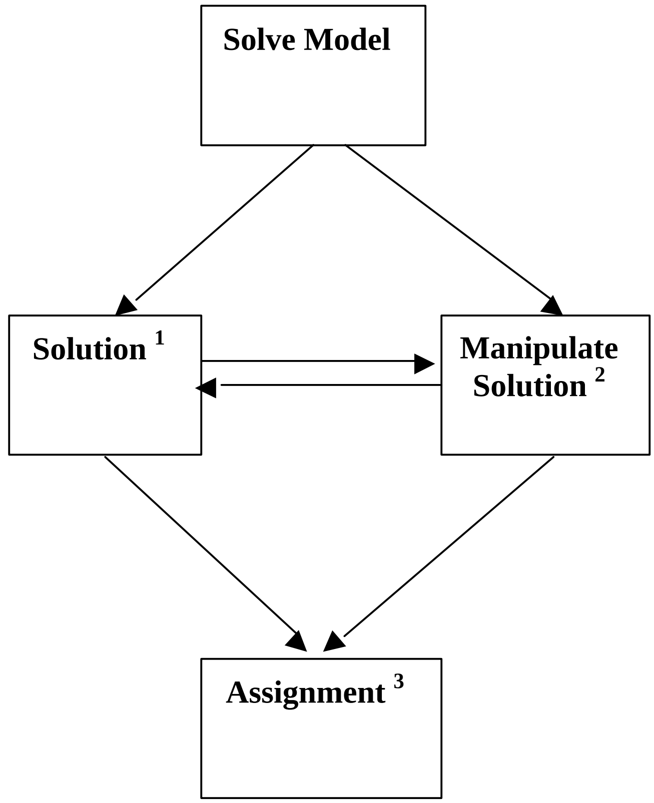




Stored Procedures for Delete Records			
#	**Name**	**Variables**	**Description**
1	ShowJobId		Returns all jobs
	DeleteJobs	JobId	Deletes a job
2	ShowJobId		Returns all jobs
3	ShowRankCodeOnJobId	JobId	Returns all the ranks for a specific job
	DeleteRankCodeOnJobRank	JobId, RankCode	Deletes a specific rank
4	ShowLanguageCodeOnJobId	JobId	Returns all the languages for a specific job
	DeleteLanguageCodeOnJobLanguage	JobId, LanguageCode	Deletes a specific language
5	ShowSpecialtyCodeOnJobId	JobId	Returns all the specialties for a specific job
	DeleteSpecialtyCodeOnJobSpecialty	JobId, SpecialtyCode	Deletes a specific specialty
6	ShowQualificationCodeOnJobId	JobId	Returns all the qualifications for a specific job

Stored Procedures for Delete Records			
#	**Name**	**Variables**	**Description**
	DeleteQualificationCodeOnJobSpecialty	JobId, QualificationCode	Deletes a specific qualification
7	ShowCredentialsIdOnJobId	JobId	Returns all the credentials for a specific job
	DeleteCredentialsIdOnJobCredentials	JobId, CredentialsId	Deletes a specific credential
8	ShowPlaceCodeOnJobId	JobId	Returns all the platforms/bases for a specific job
	DeletePlaceCodeOnJobPlace	JobId, PlaceCode	Deletes a specific base/platform
9	ShowApplicantIdLastNameFirstName		Returns all officers’ last, first name and rank
	DeleteApplicants	ApplicantId	Deletes an officer
10	ShowApplicantIdLastNameFirstNameWORank		Returns all officers’ last and first name
11	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns an officer’s first name, last name and rank
	ShowLanguageCodeOnApplicantId	ApplicantId	Returns the languages and grades of a specific officer
	DeleteApplicantIdOnApplicantLanguage	ApplicantId, LanguageCode	Deletes an officer’s language and grade
12	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns an officer’s first name, last name and rank
	ShowCredentialsIdOnApplicantId	ApplicantId	Returns the credentials and grades of a specific officer
	DeleteApplicantIdOnApplicantCredentials	ApplicantId, CredentialsId	Deletes an officer’s credential and grade
13	ShowApplicantIdLastNameFirstNameRankNameOnApplicantId	ApplicantId	Returns an officer’s first name, last name and rank
	ShowQualificationCodeOnApplicantId	ApplicantId	Returns the qualifications and grades of a specific officer
	DeleteApplicantIdOnQualificationApplicant	ApplicantId, QualificationCode	Deletes an officer’s qualification and grade
14	ShowCredentialsId		Returns all the credentials
	DeleteCredentials	CredentialsId	Deletes a credential
15	ShowPlaceCode		Returns all the platforms /bases
	DeletePlaces	PlaceCode	Deletes a platform /base
16	ShowCommandCode		Returns all the commands
	DeleteCommands	CommandCode	Deletes a command
17	ShowQualificationCode		Returns all the qualifications
	DeleteQualifications	QualificationCode	Deletes a qualification
18	ShowSpecialtyCode		Returns all the specialties
	DeleteSpecialties	SpecialtyCode	Deletes a specialty
19	ShowLanguageCode		Returns all the languages
	DeleteLanguages	LanguageCode	Deletes a language
20	ShowRankCode		Returns all the ranks
	DeleteRanks	RankCode	Deletes a rank
21	ShowCoefficients		Returns all the coefficients
	DeleteCoefficient	WeightName	Deletes a coefficient

e. Solve Model



Stored Procedures for Solve Model		
#	**Name**	**Variables**
1	dec_CheckHValueExists	Counter
	dec_CheckHValueNotNull	JobId, PlaceCode, ApplicantId
	dec_ComputeMaxValue	Counter
	dec_ComputeMeanValue	
	dec_COUNTER_Fill	
	dec_CountPriorityRecords	
	dec_Credentials	JobId, ApplicantId
	dec_Credentials1	ApplicantId, CredentialsId
	dec_Credentials2	JobId, CredentialsId
	dec_Experience	JobId, ApplicantId
	dec_H_Fill	
	dec_H_Function	JobId, ApplicantId, PlaceCode
	dec_H_Normalize	
	dec_Language	JobId, ApplicantId
	dec_Language1	ApplicantId, LanguageCode
	dec_Language2	JobId, LanguageCode
	dec_Main	
	dec_MAX_VALUE_Fill	

Stored Procedures for Solve Model		
#	Name	Variables
	dec_PreferenceApplicantReturn	JobId, ApplicantId, PlaceCode
	dec_PreferenceCommandReturn	JobId, ApplicantId, PlaceCode
	dec_PRIORITY_Fill	
	dec_QualificationExists1	ApplicantId, QualificationCode
	dec_QualificationExists2	JobId, QualificationCode
	dec_Qualifications	JobId, ApplicantId
	dec_Rank	JobId, ApplicantId
	dec_RankExists1	ApplicantId, RankCode
	dec_RankExists2	JobId, RankCode
	dec_SetMAXValueNull	Counter
	dec_ShowDeletedJobs	
	dec_ShowJobNameOnJobId	JobId
	dec_ShowSolution	
	dec_ShowUnassignedApplicants	
	dec_Specialty	JobId, ApplicantId
	dec_SpecialtyExists1	ApplicantId, SpecialtyCode
	dec_SpecialtyExists2	JobId, SpecialtyCode
	dec_UNASSIGNED_APPLICANTS_Fill	
2	dec_Delete_Job_Manipulate	JobId, PlaceCode
	dec_DELETED_JOBS_MANIPULATE_DeleteRecord	JobId, PlaceCode
	dec_DELETED_JOBS_MANIPULATE_Fill	
	dec_DeleteEmptyJobs	
	dec_DeleteJob	
	dec_DeleteJobUsedValues	Counter
	dec_EstimateFunction	
	dec_FindMaxValue	JobId, PlaceCode, MAXValue
	dec_MANIPULATE_SOLUTION_Fill	
	dec_MANIPULATE_SOLUTION_InsertRecord	JobId, PlaceCode, ApplicantId
	dec_MAX_VALUE_ALL_JOBS_Fill	
	dec_ShowDeletedJobsManipulate	
	dec_ShowEstimateFunctionResult	
	dec_ShowJobNameOnJobId	JobId
	dec_ShowManipulateSolution	
	dec_ShowNotNullHValue	
	dec_ShowPlaceNameOnPlaceCode	PlaceCode
	dec_ShowUnassignedApplicantsManipulate	
	dec_UNASSIGNED_APPLICANTS_MANIPULATE_DeleteRecord	ApplicantId
	dec_UNASSIGNED_APPLICANTS_MANIPULATE_Fill	
3	AcceptSolutionFromMAXTable	
	AcceptSolutionFromManipulateSolutionTable	

D. USE CASES

This section describes examples of use cases. Each of these use cases is a sequence of actions the three categories of users have to perform. The following lines present a sequence of screens that each user goes through while the user performs his/her basic roles.

1. Officer

The basic functionalities the officer has to do are to delete a preference he has already selected and add a new preference.

a. Delete a Preference

(1) The Officer Logs In.

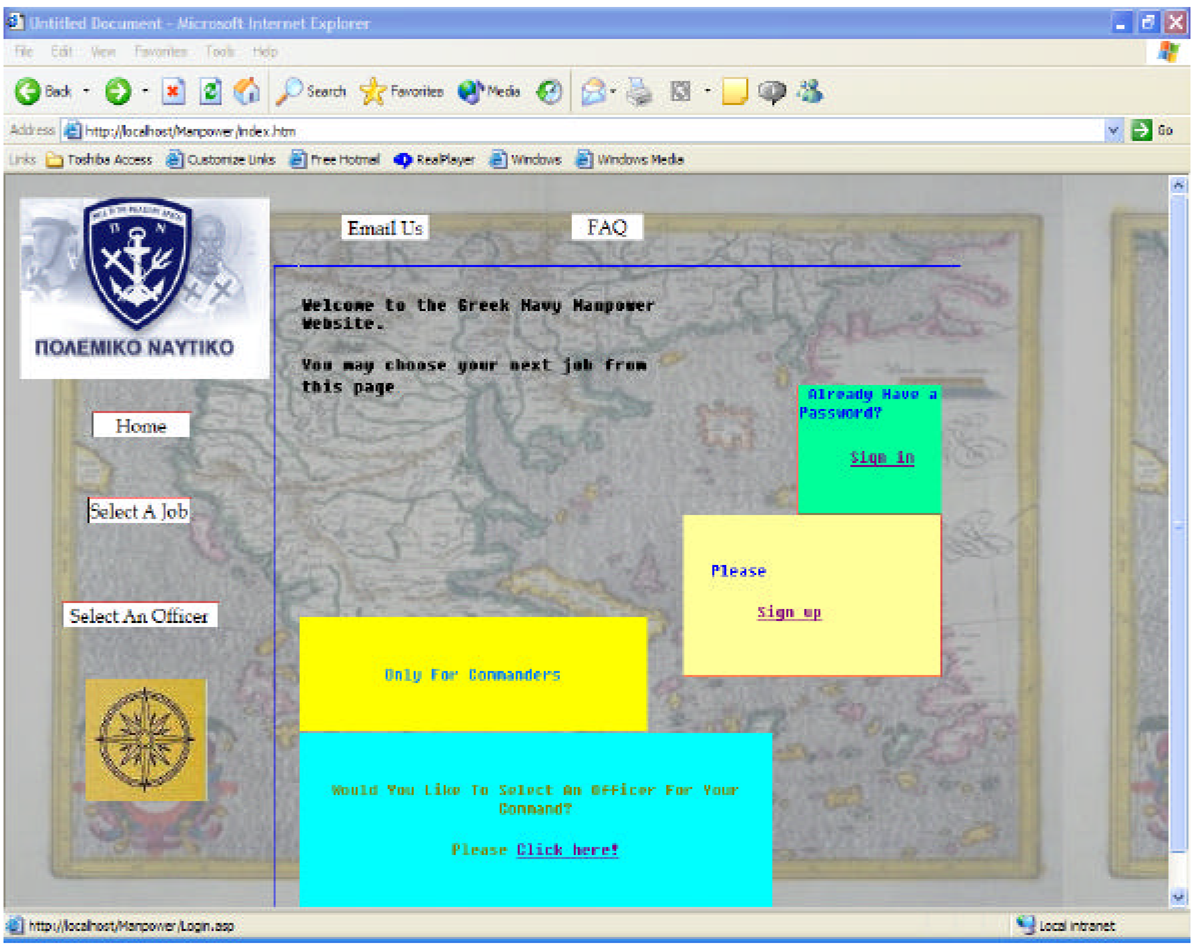


Figure 64. The Officer Selects the 'Already Have a Password? Sign In'-Manpower Website.

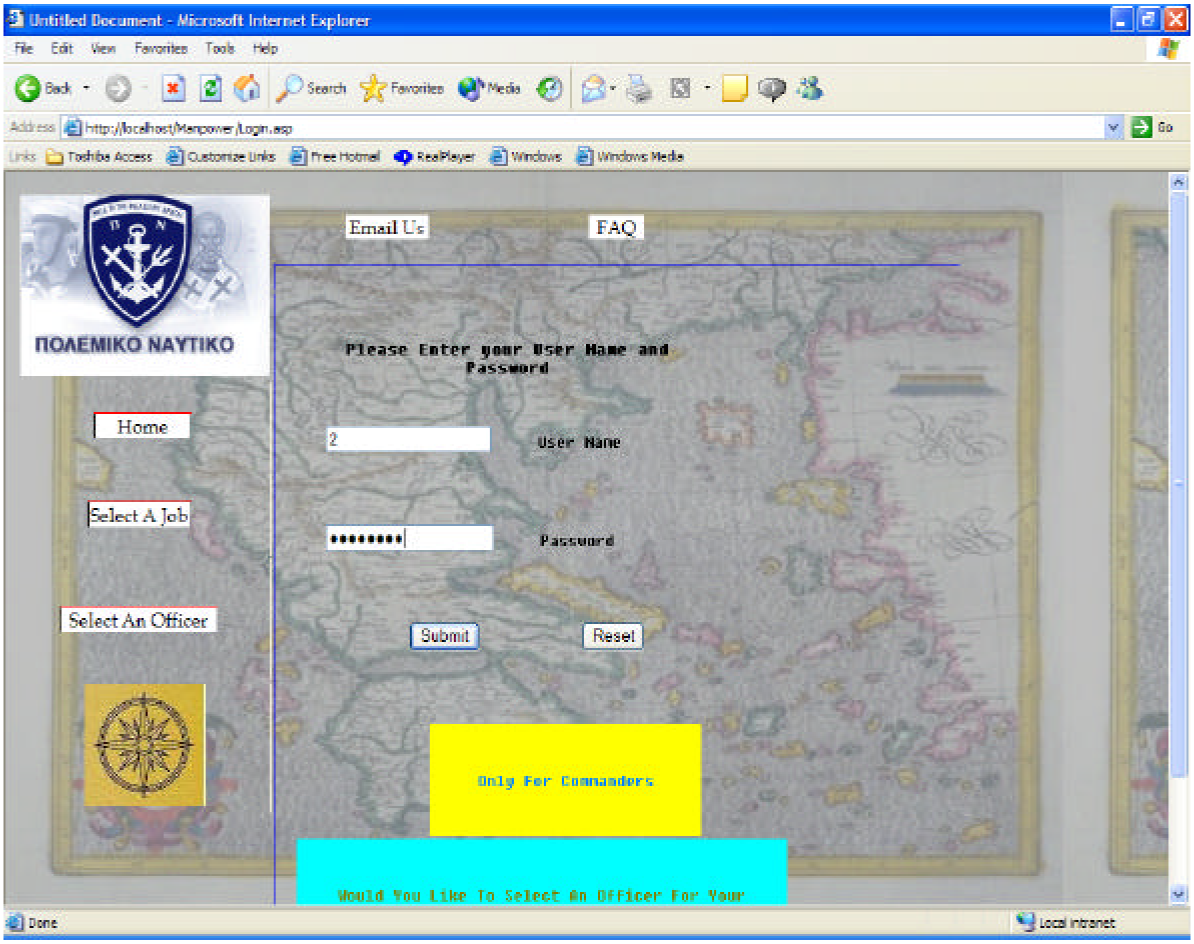


Figure 65. The Officer Types the User Name and Password-Manpower Website.

(2) The Officer Deletes a Preference

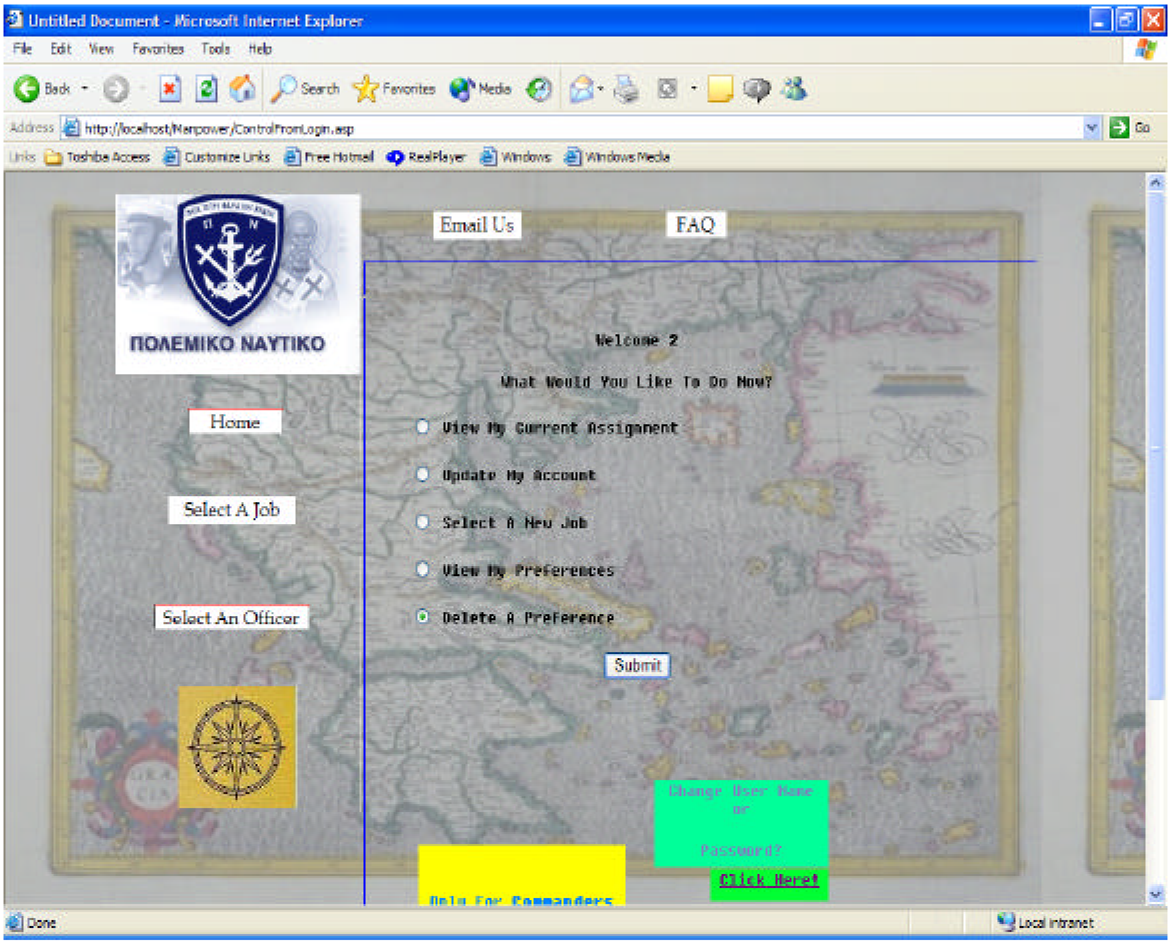


Figure 66. The Officer Selects ‘Delete A Preference’-Manpower Website.

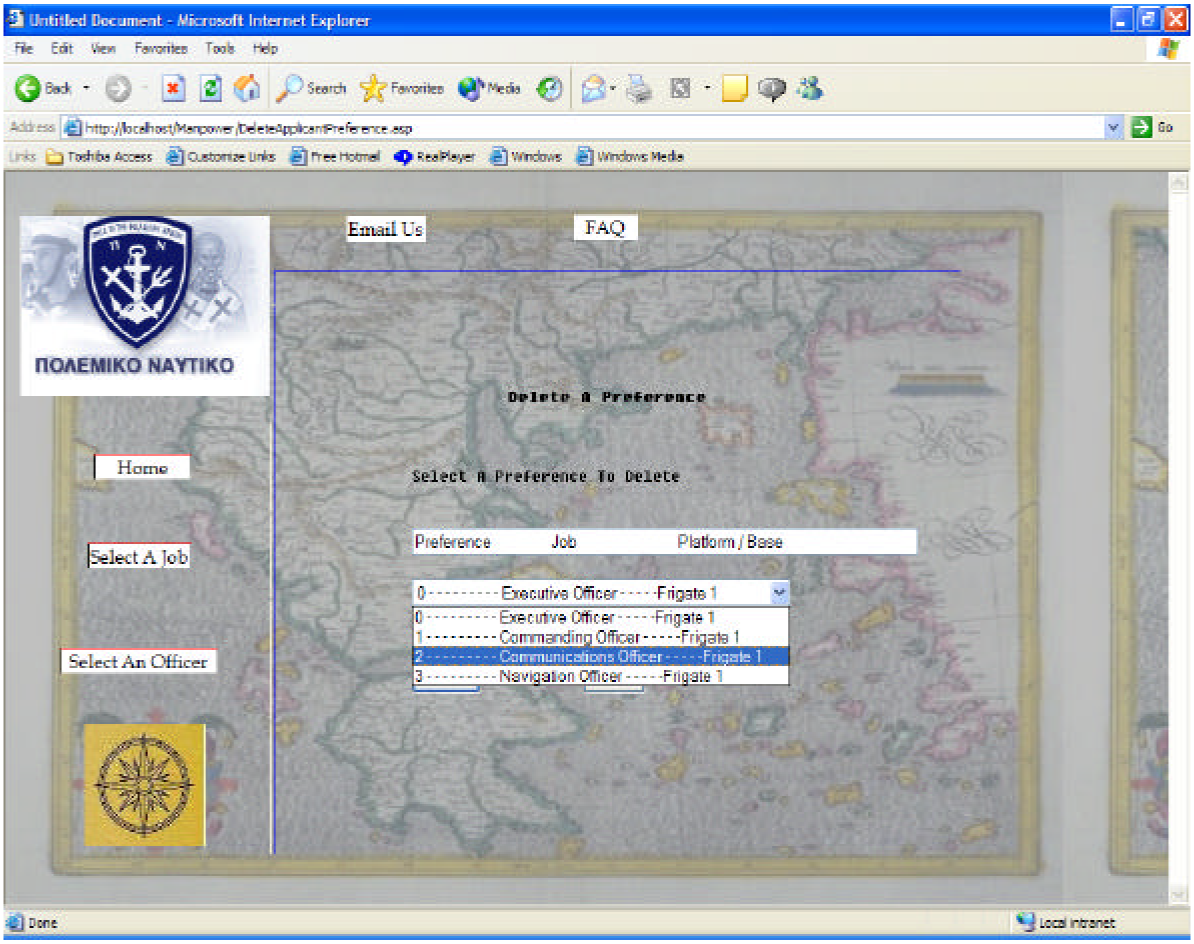


Figure 67. The Officer Selects Preference Number 2 to Delete-Manpower Website.

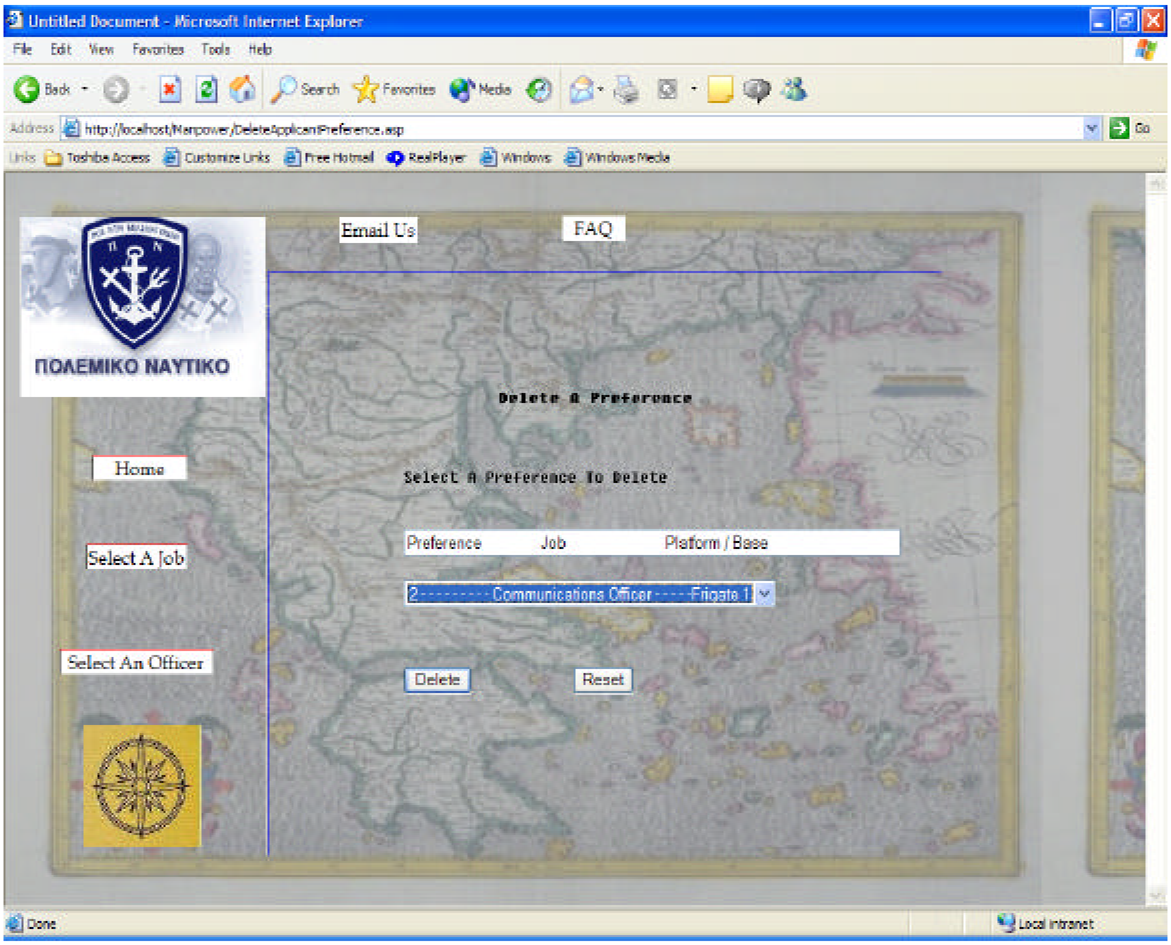


Figure 68. Preference Number 2 is Selected-Manpower Website.

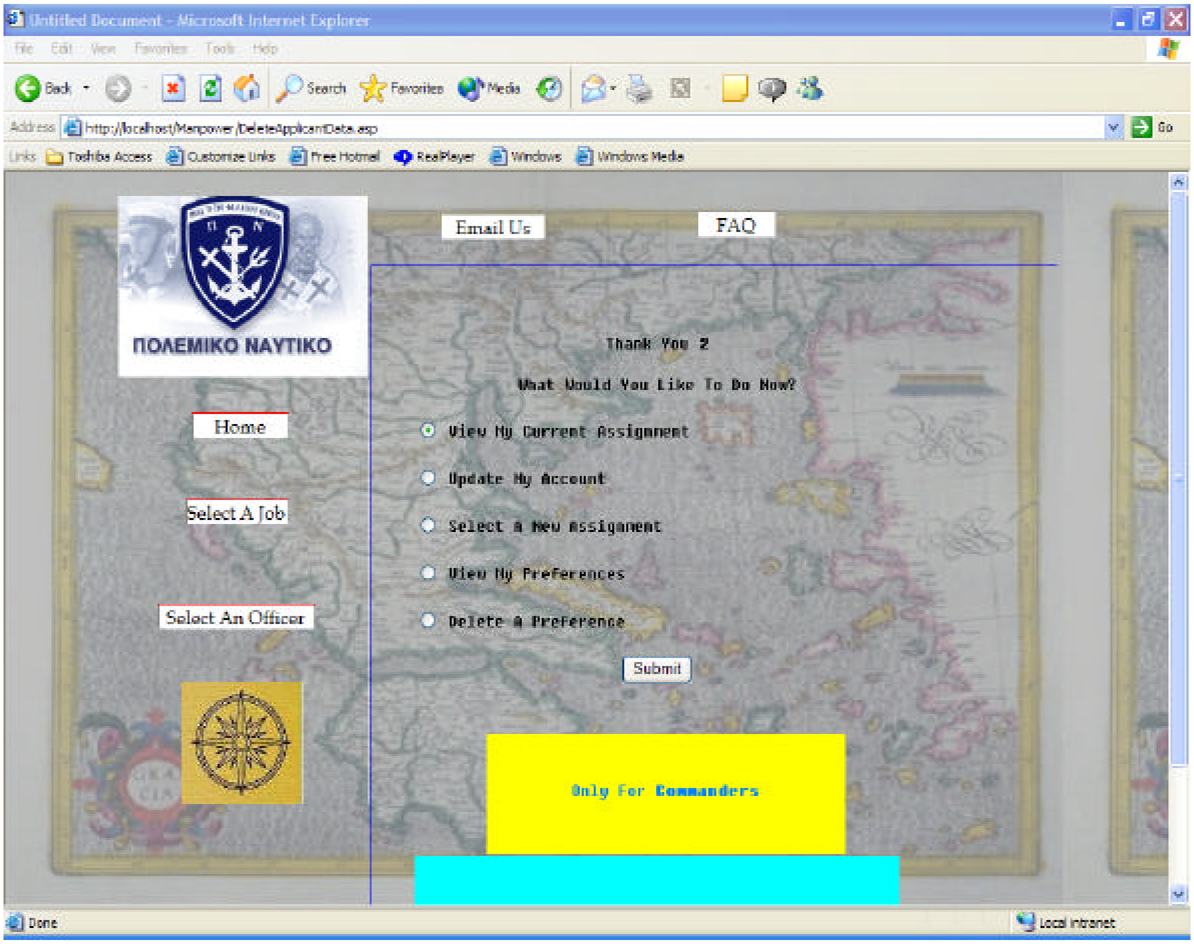


Figure 69. Preference Number 2 is Deleted and the Officer Goes Back to the Control Page-Manpower Website.

b. Add a Preference

(1) The Officer Logs in the Same Manner As Described Above.

(2) The Officer Adds a Preference

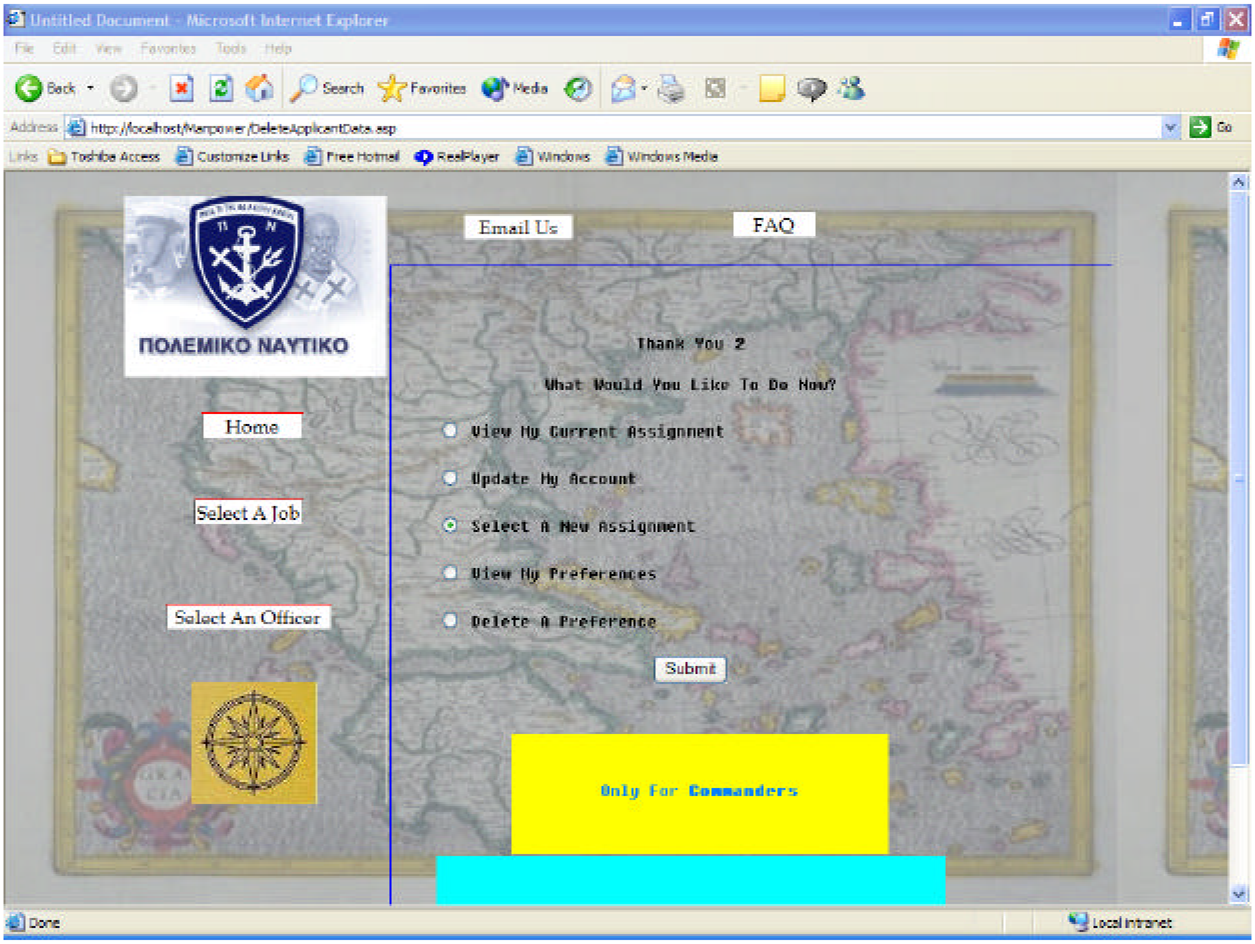


Figure 70. The Officer Selects the 'Select A New Assignment' Option-Manpower Website.

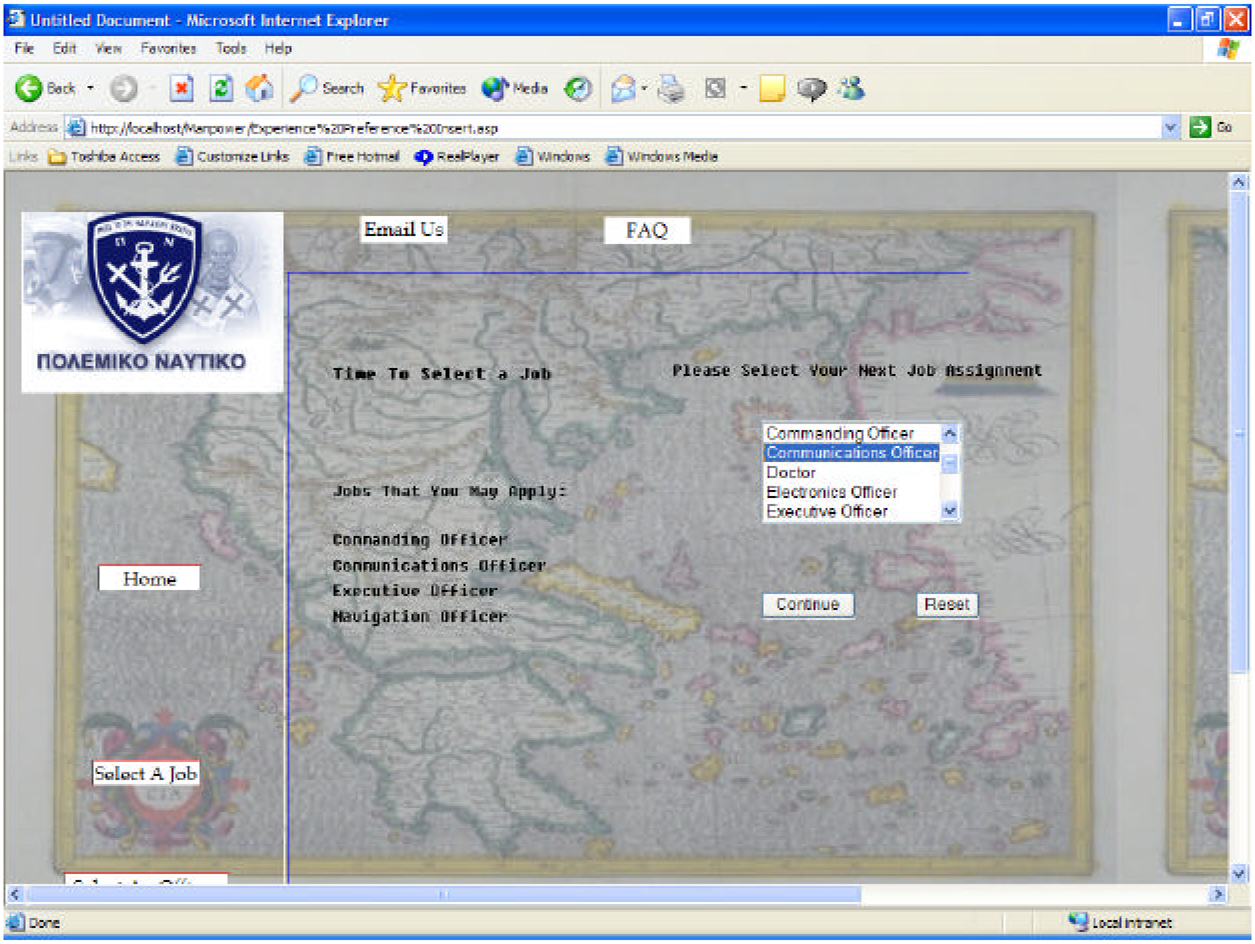


Figure 71. The Officer Selects the Communications Officer-Manpower Website.

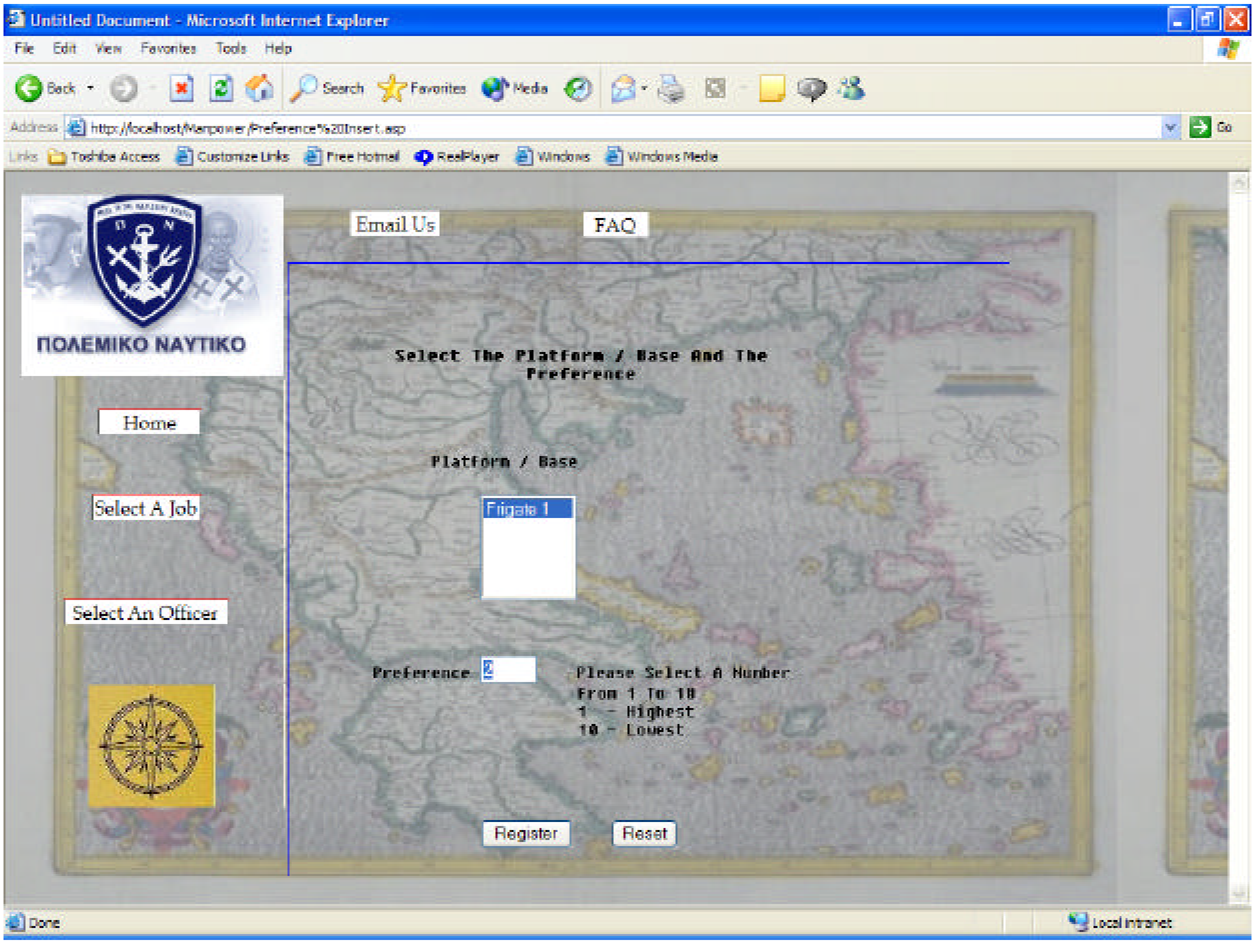


Figure 72. The Officer Selects the Frigate 1 and Preference 2-Manpower Website.

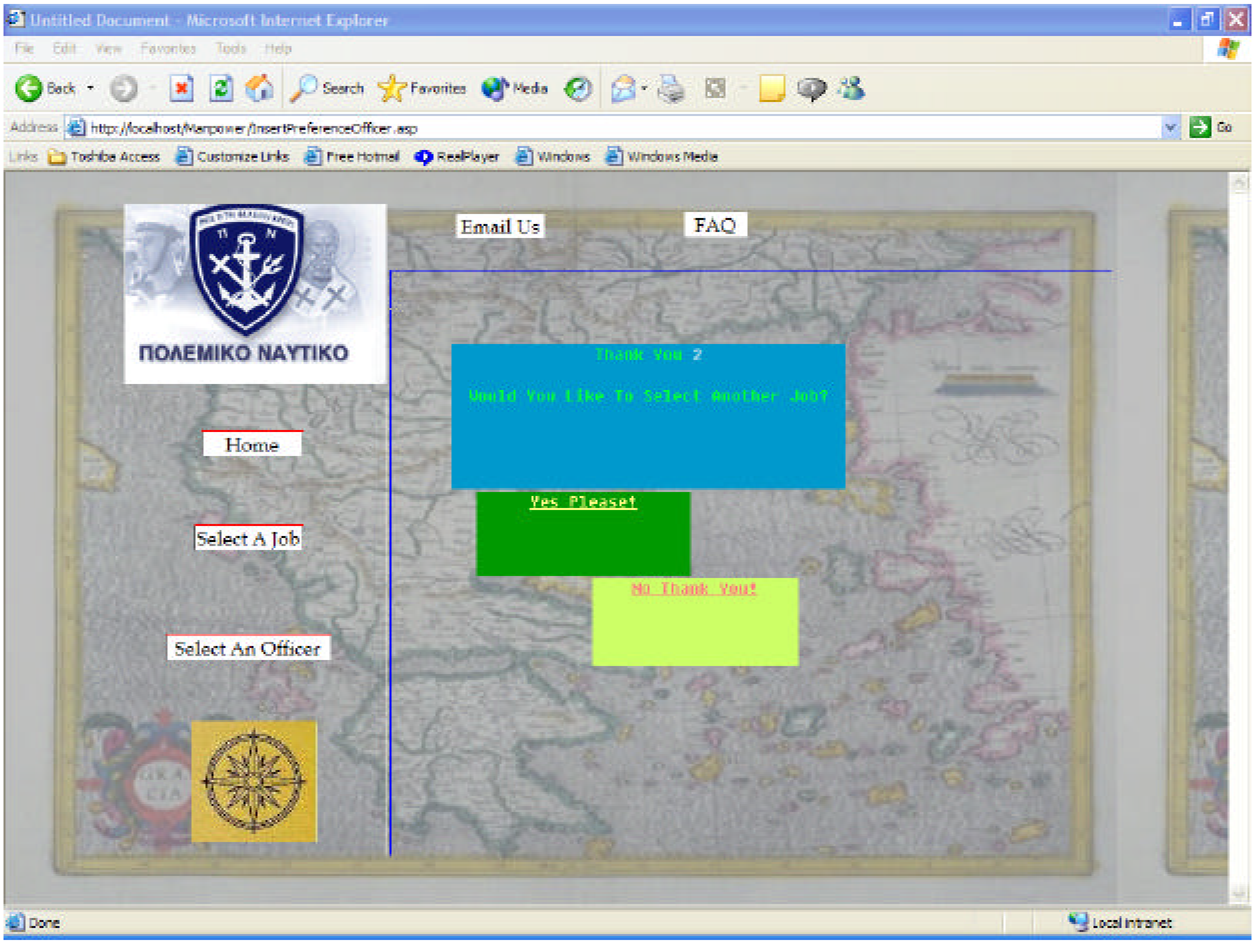


Figure 73. The Officer Has Applied His/Her Preference-Manpower Website.

2. Command

The basic functionalities the command has to do are to delete a preference it has already selected and add a new preference.

a. Delete a Preference

(1) Log In. The command logs in the same way the officer does but the command selects the 'Would You Like To Select An Officer For Your Command? Please Click here!' option instead.

(2) Delete a Preference.

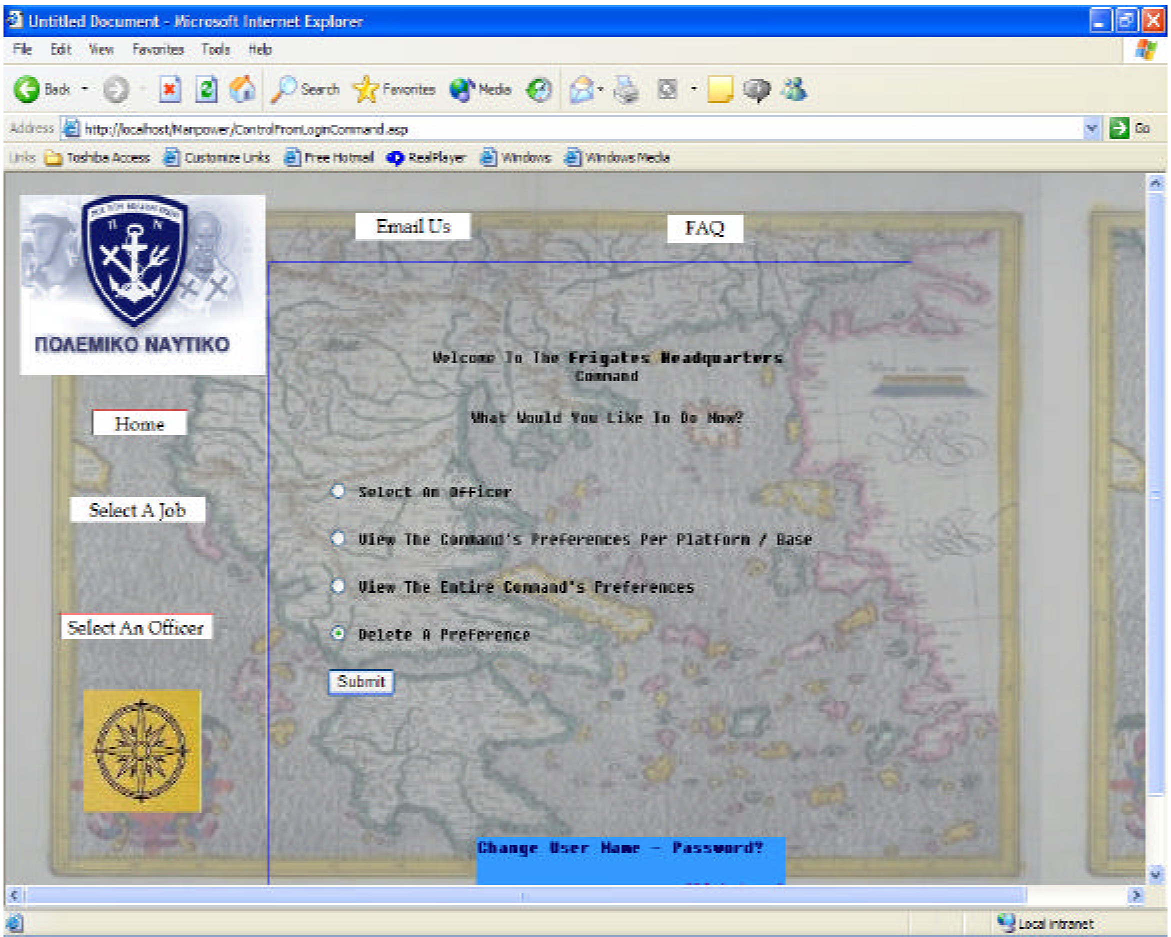


Figure 74. The Command Selects 'Delete A Preference'-Manpower Website.

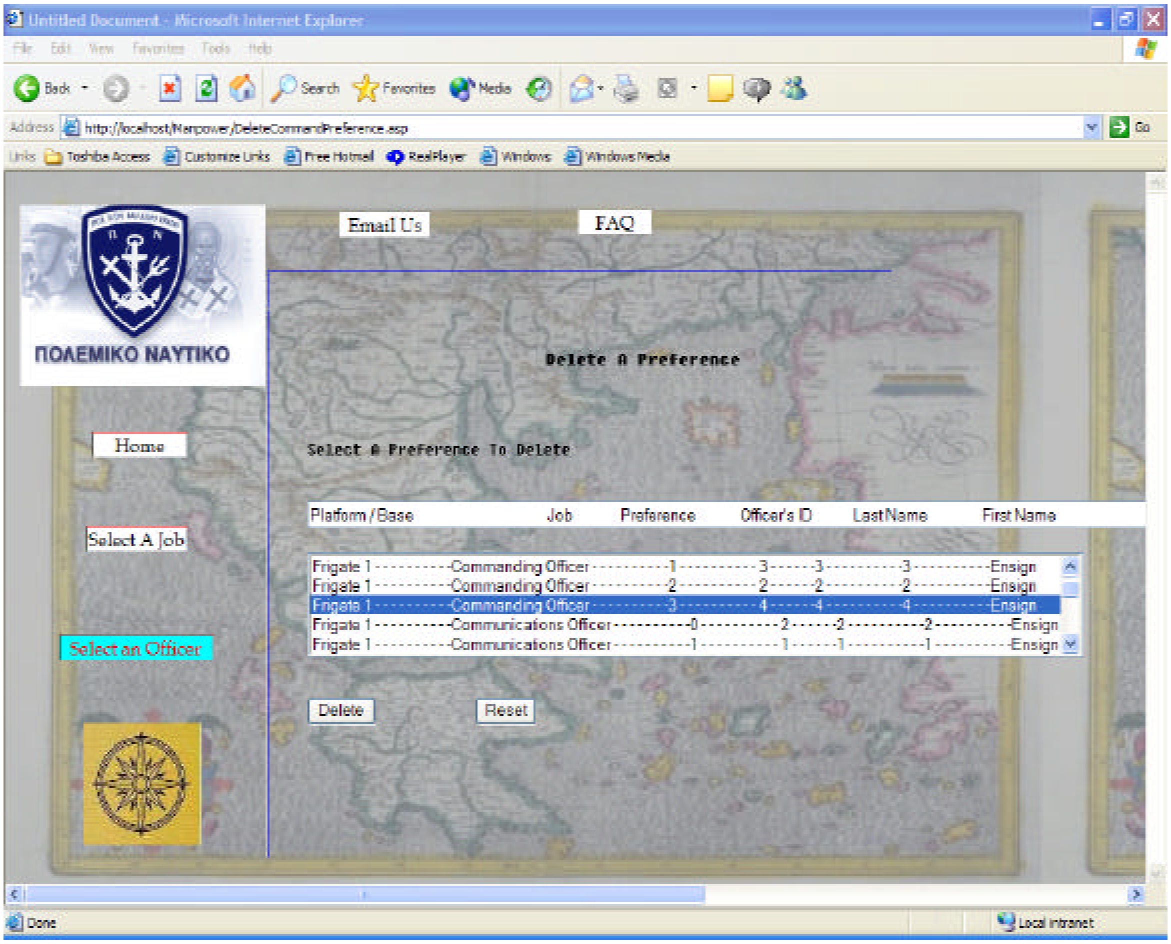


Figure 75. The Command Selects the job Commanding Officer for Frigate 1 with Preference Number 3-Manpower Website.

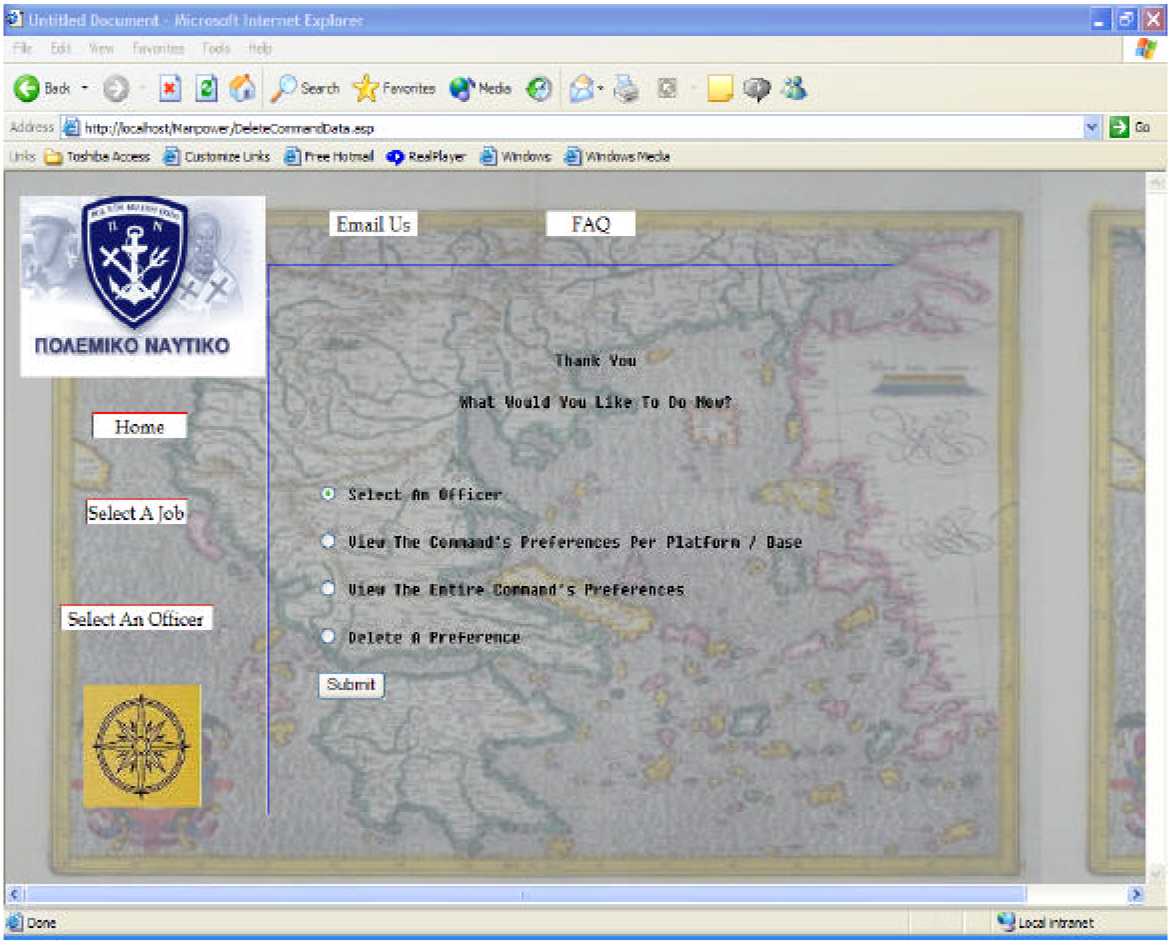


Figure 76. The Preference Number 3 is Deleted-Manpower Website.

b. Add a Preference

(1) The Command Logs In As Described Above.

(2) The Command Adds a Preference.

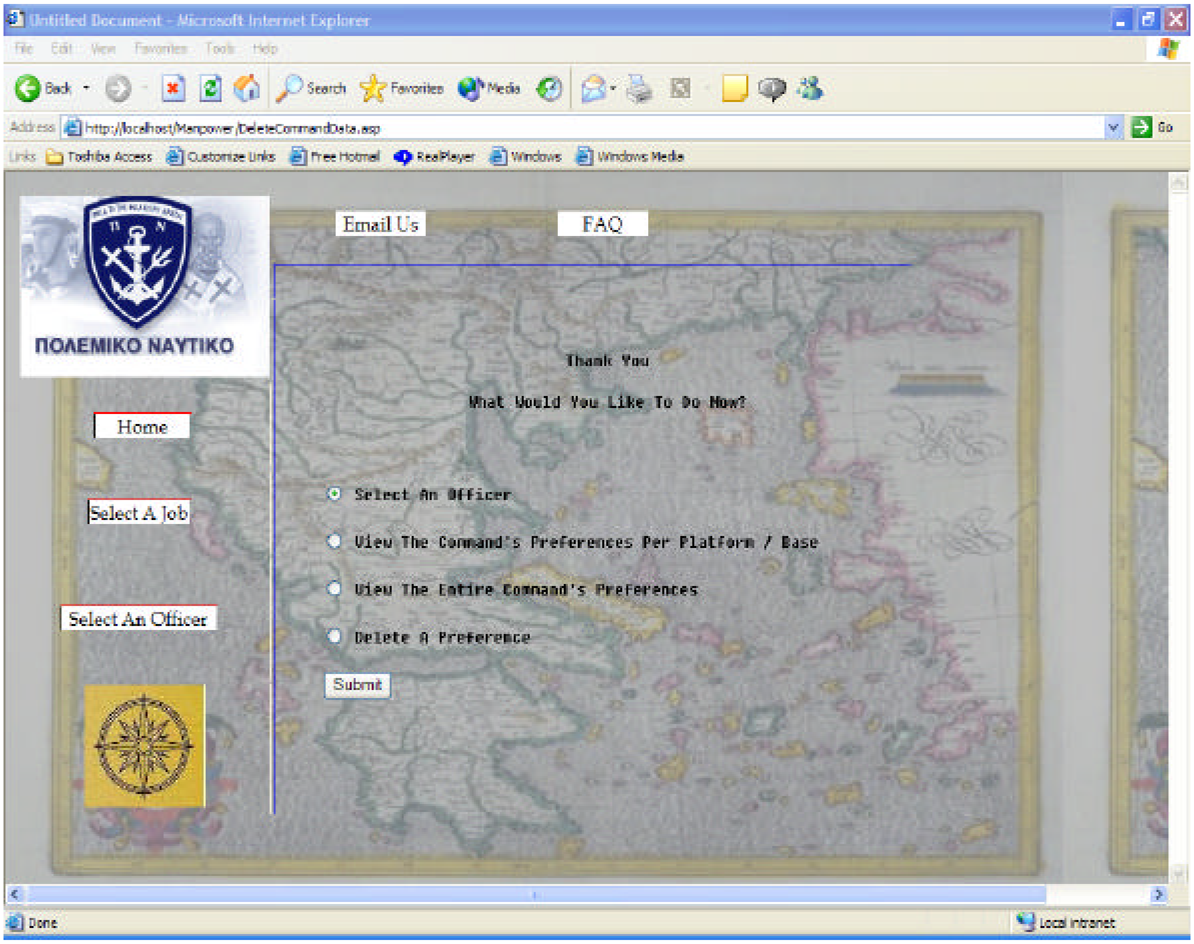


Figure 77. The Command Selects the 'Select An Officer' Option-Manpower Website.

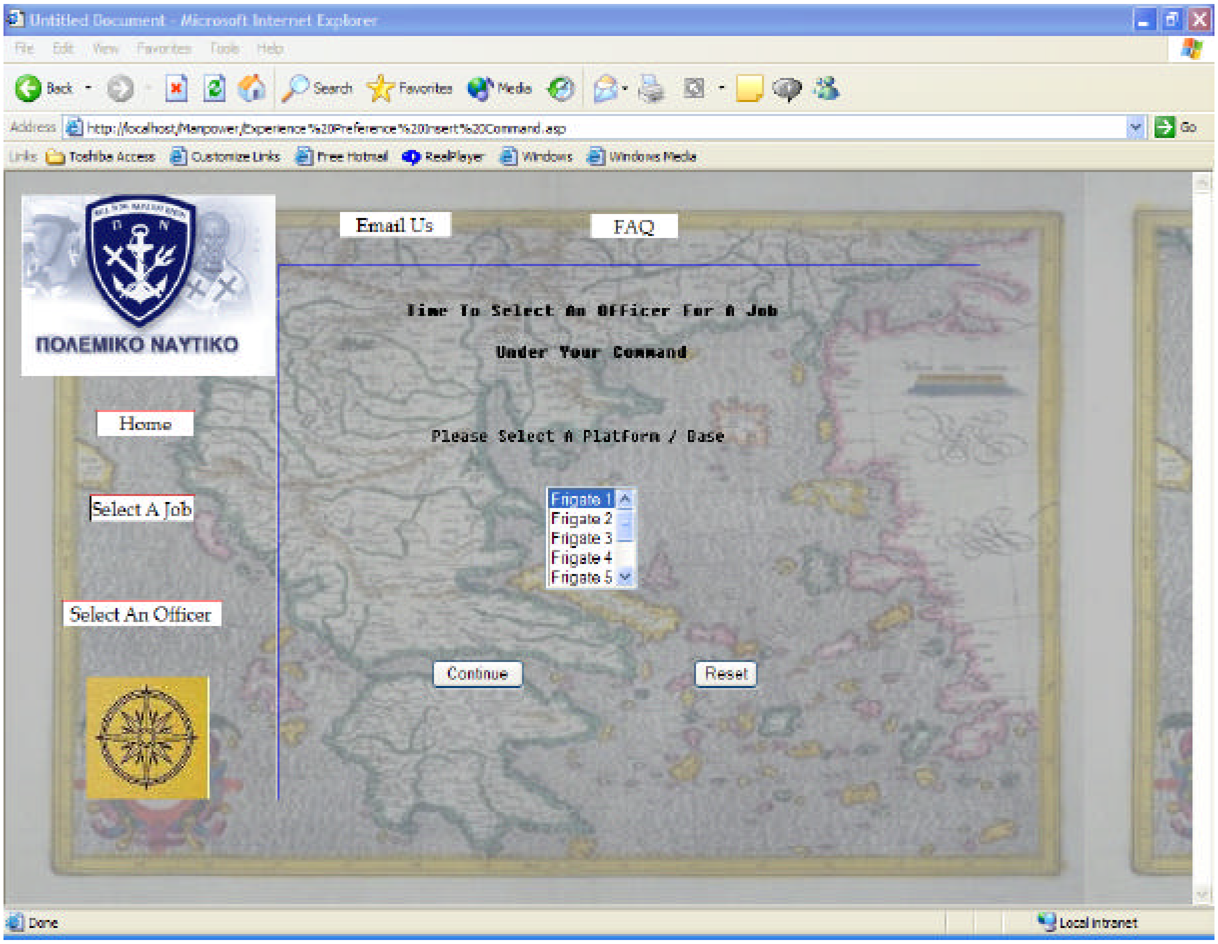


Figure 78. The Command Selects Frigate 1-Manpower Website.

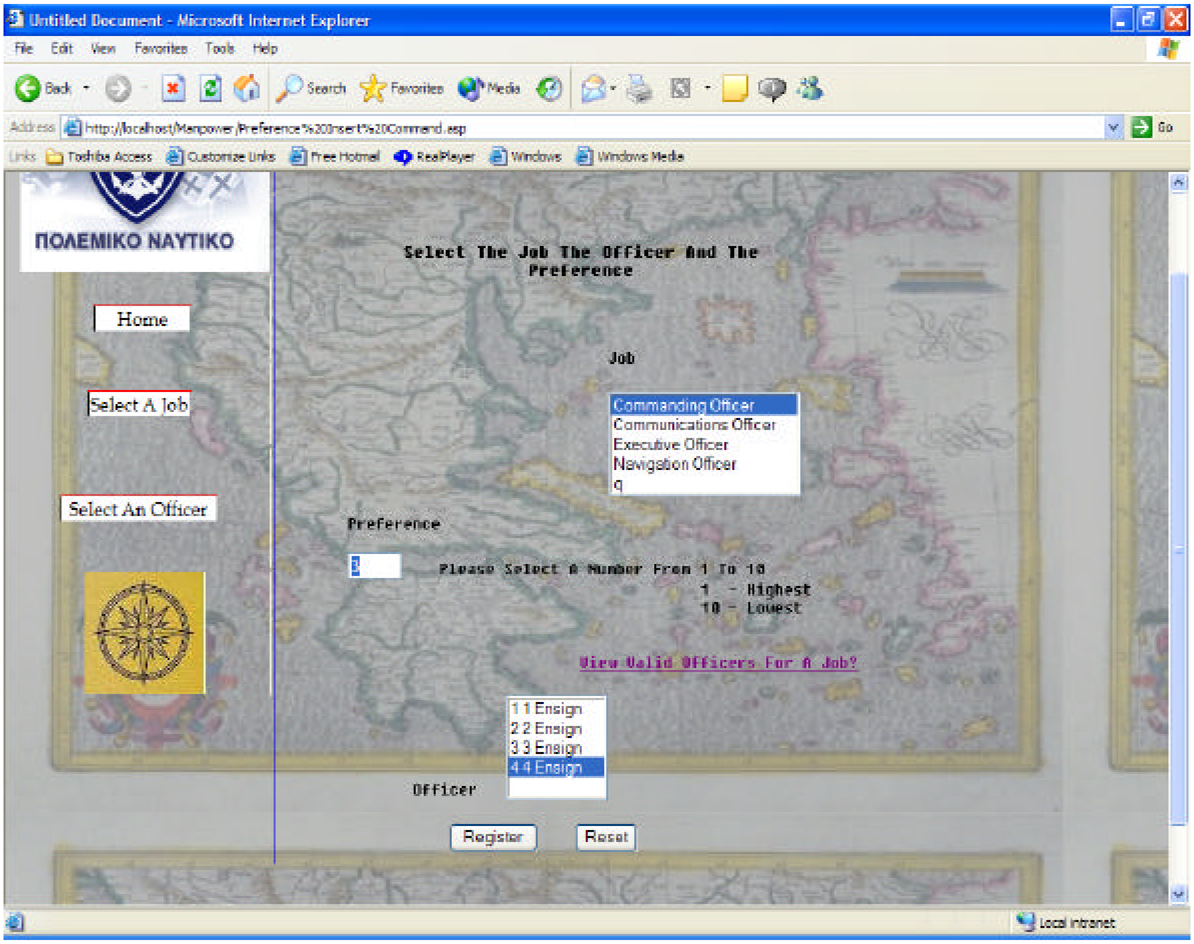


Figure 79. The Command Selects the Commanding Officer Job and Officer 4 with Preference Number 3-Manpower Website.

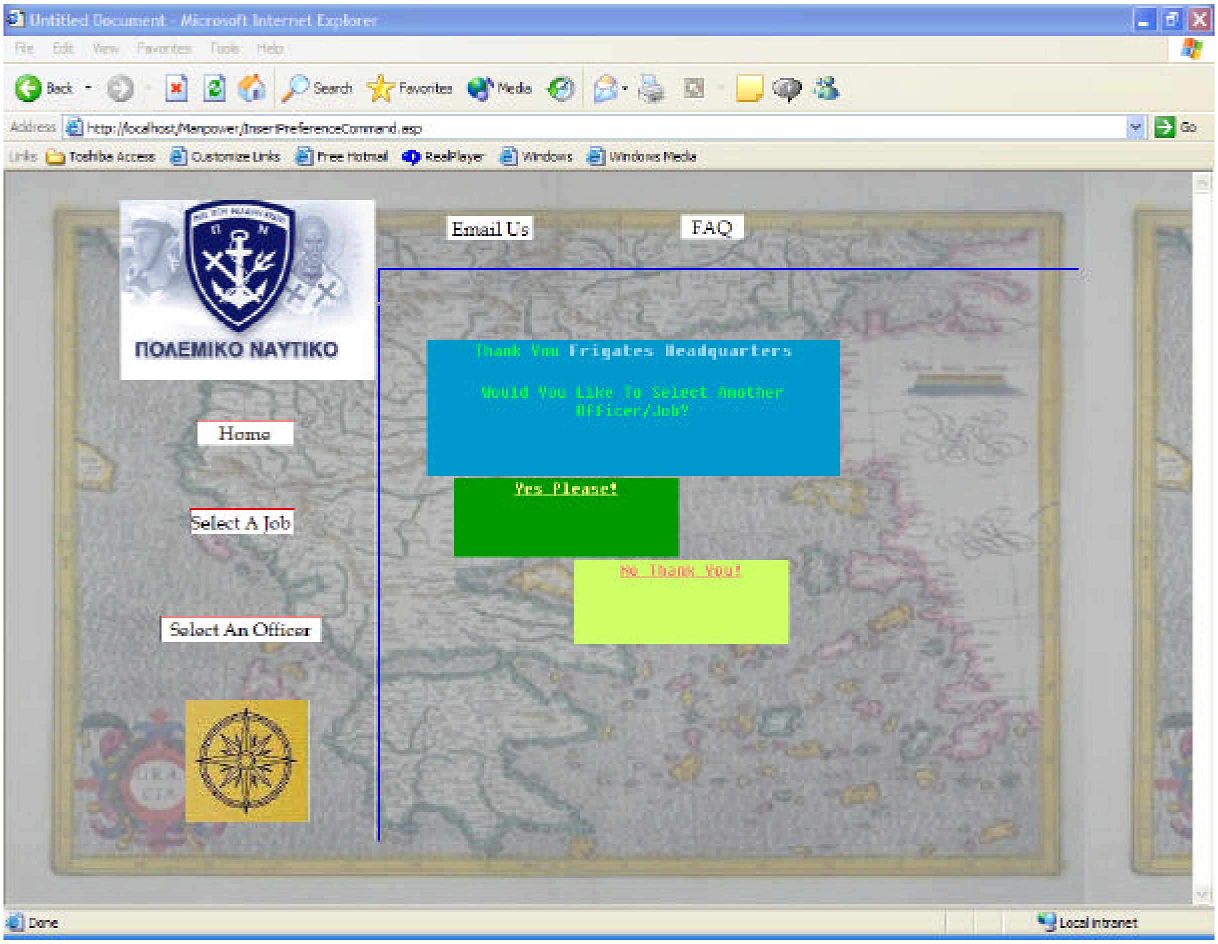


Figure 80. The Commanding Officer Job and Officer 4 with Preference Number 3 Is Selected-Manpower Website.

3. Detailer

The main job for the detailer is to solve the multi-criteria model and make any changes if the detailer wishes to.

a. Solve the Model

(1) The Detailer Has to Log In First.

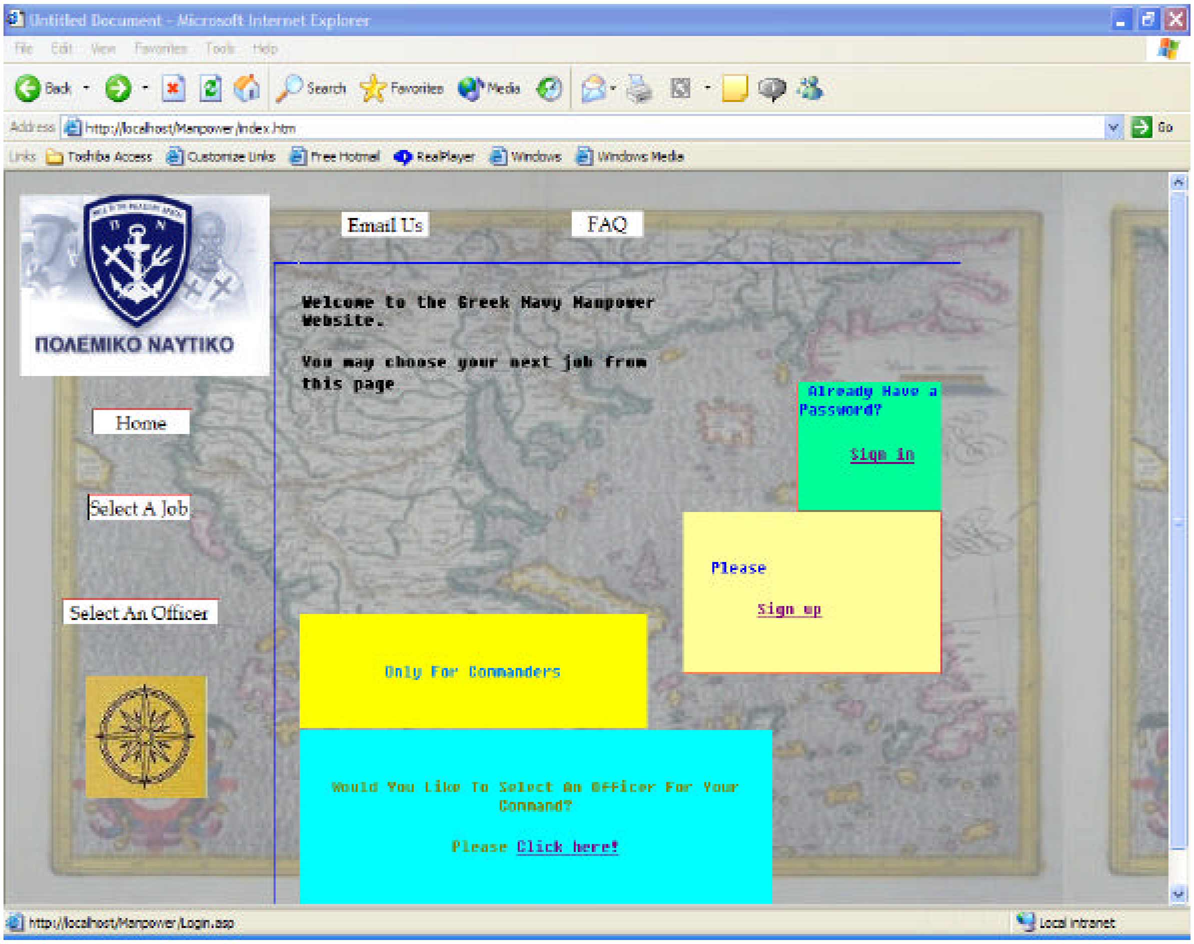


Figure 81. The Detailer Selects the 'Already Have a Password? Sign In'-Manpower Website.

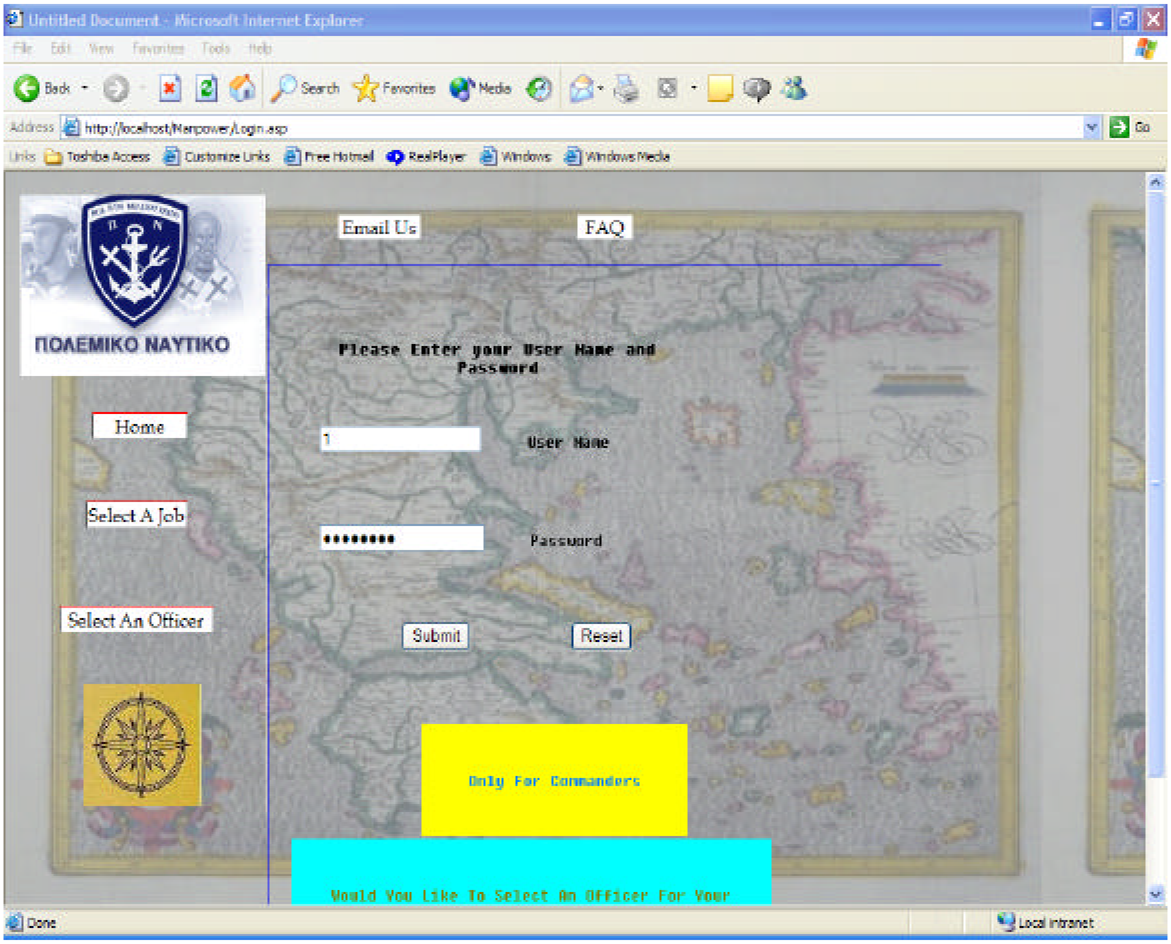


Figure 82. The Detailer Types the User Name and Password-Manpower Website.

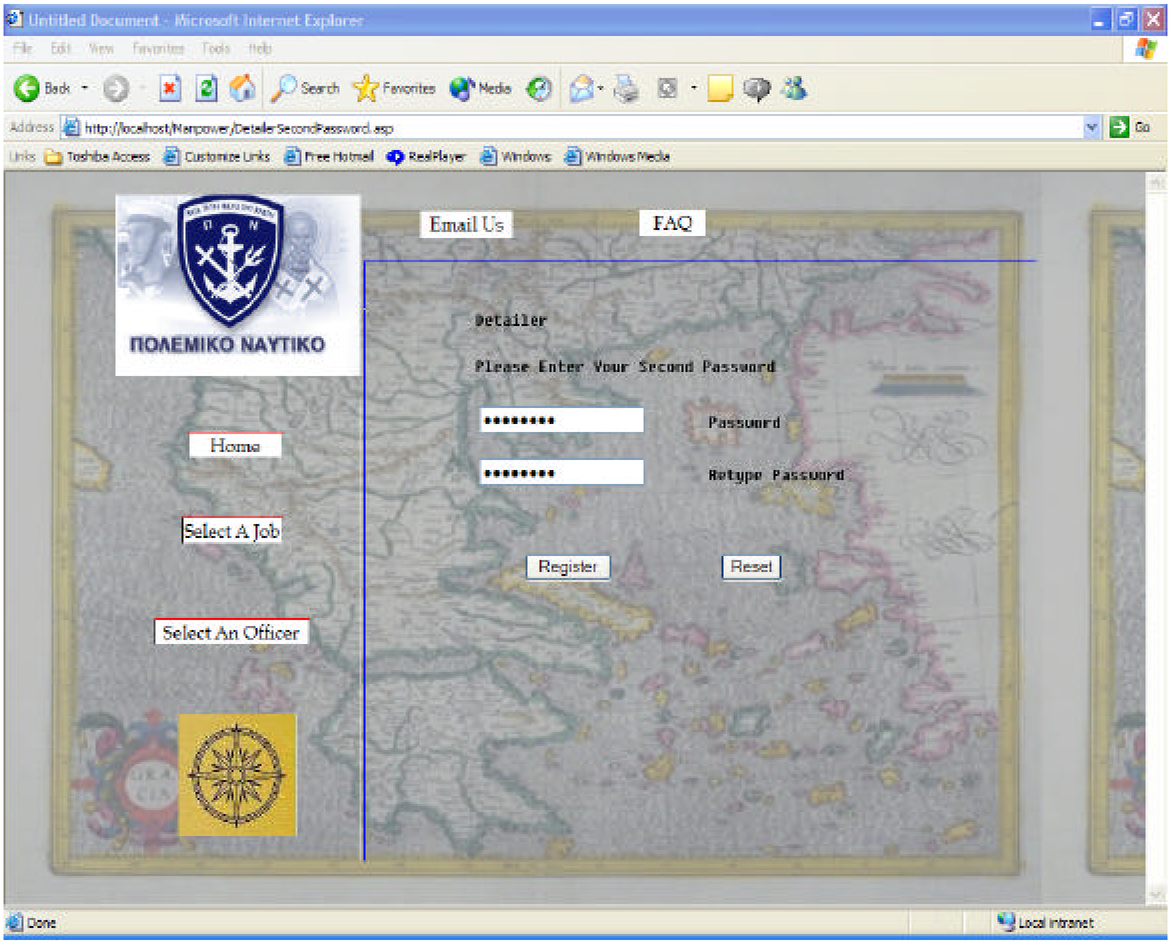


Figure 83. The Detailer Types the Second Password the Detailer Has-Manpower Website.

(2) The Detailer Solves the Model.

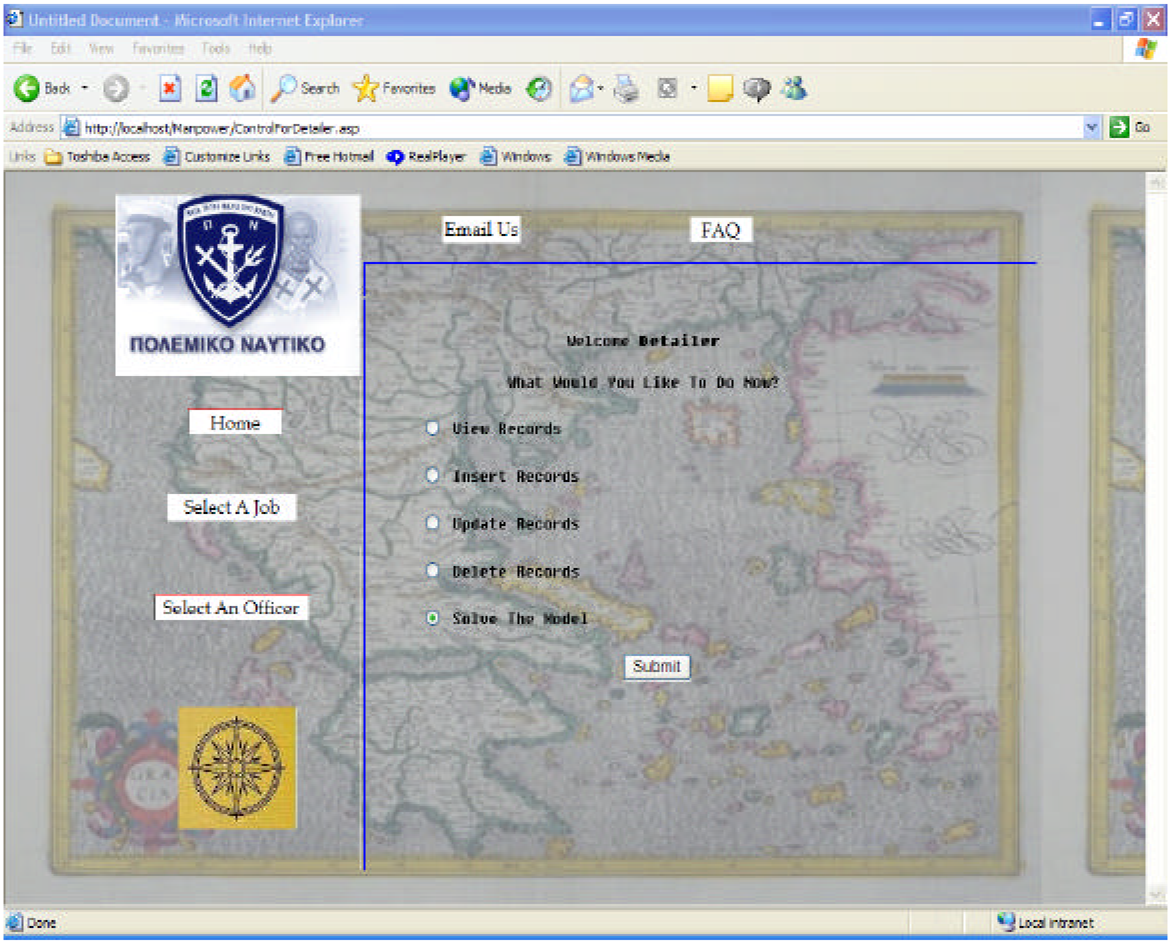


Figure 84. The Detailer Selects the 'Solve The Model' Option-Manpower Website.

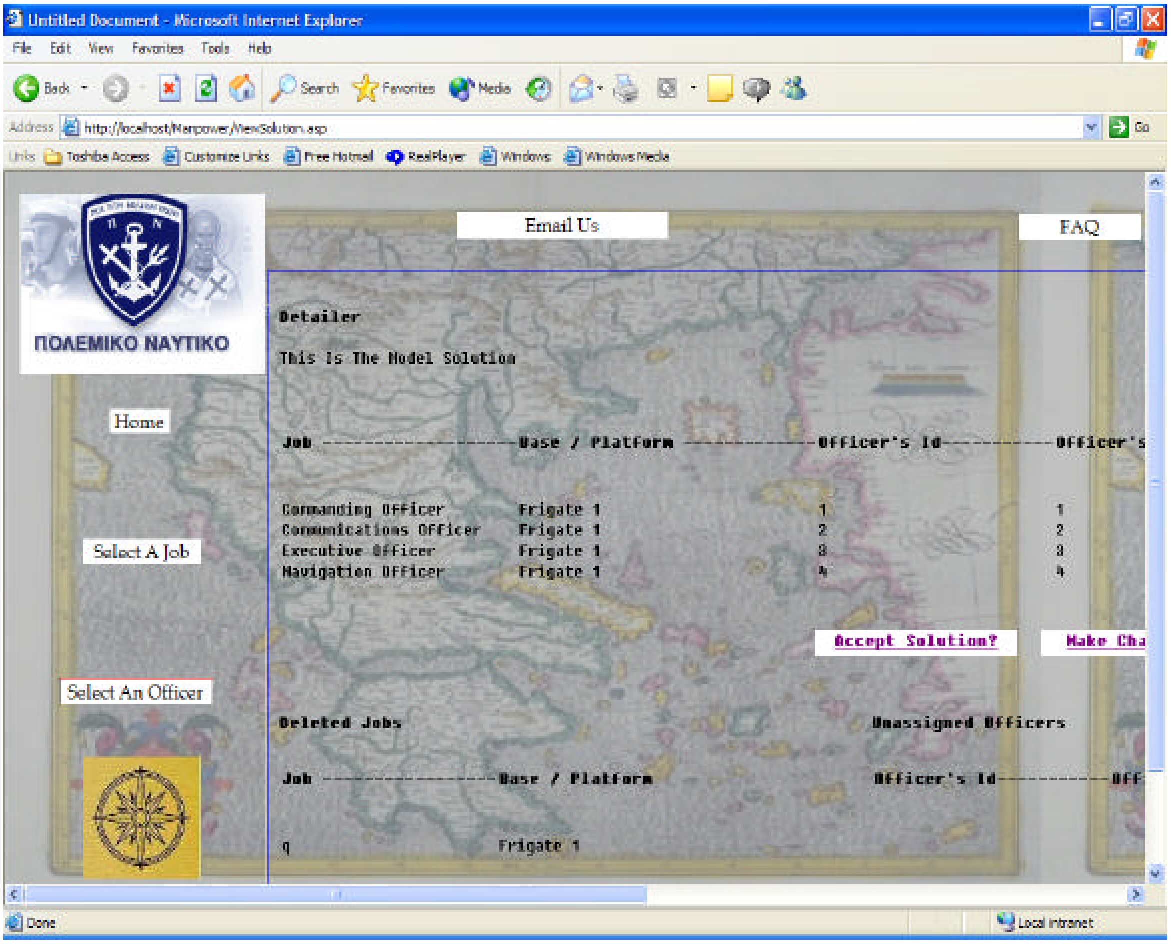


Figure 85. The Algorithm Solution (Screen 1)-Manpower Website.

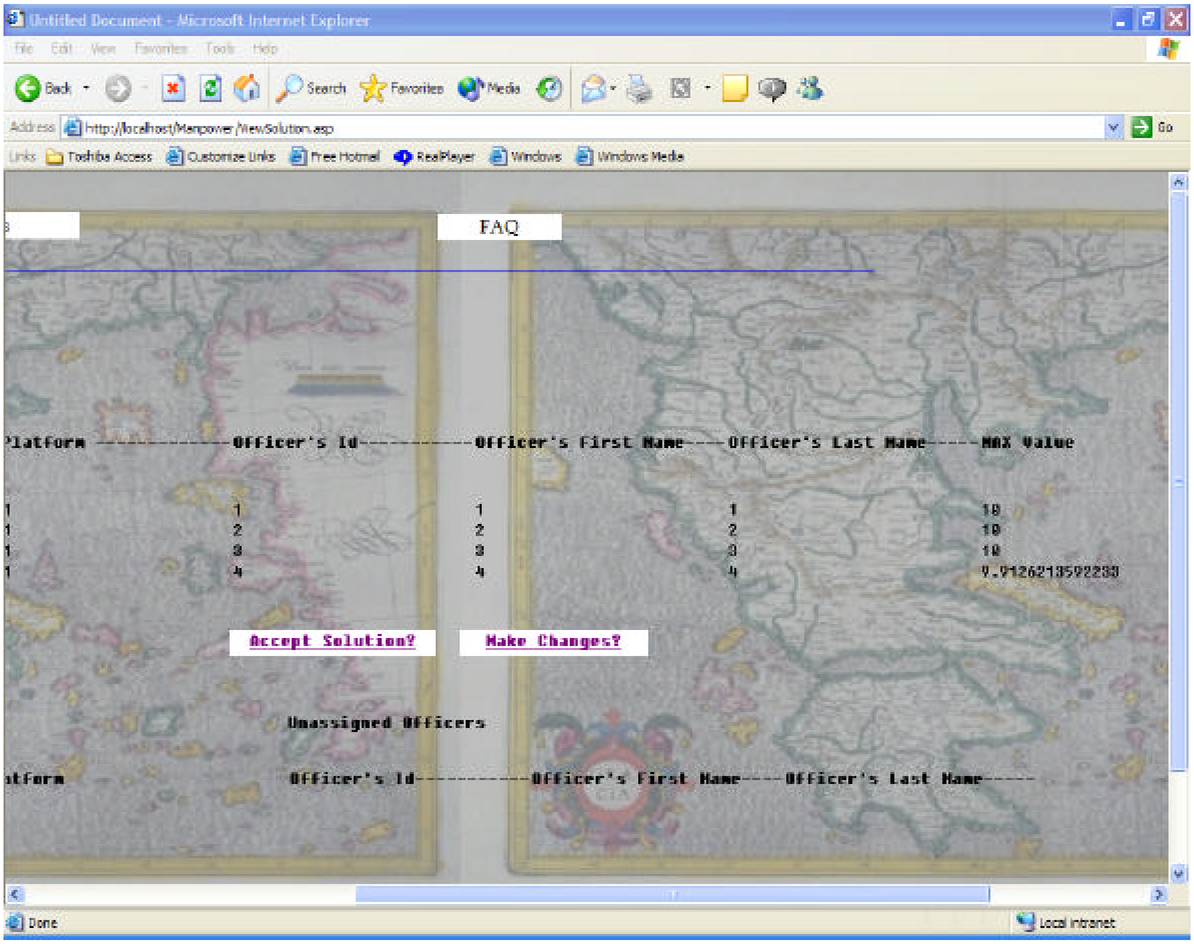


Figure 86. The Algorithm Solution (Screen 2)-Manpower Website.

(3) The Detailer Makes Changes. In Figure 37, the detailer selects the 'Make Changes' option. The page that follows allows the detailer to wipe out a job and an officer from the solution set, by selecting the MAX Value link that corresponds to that job.

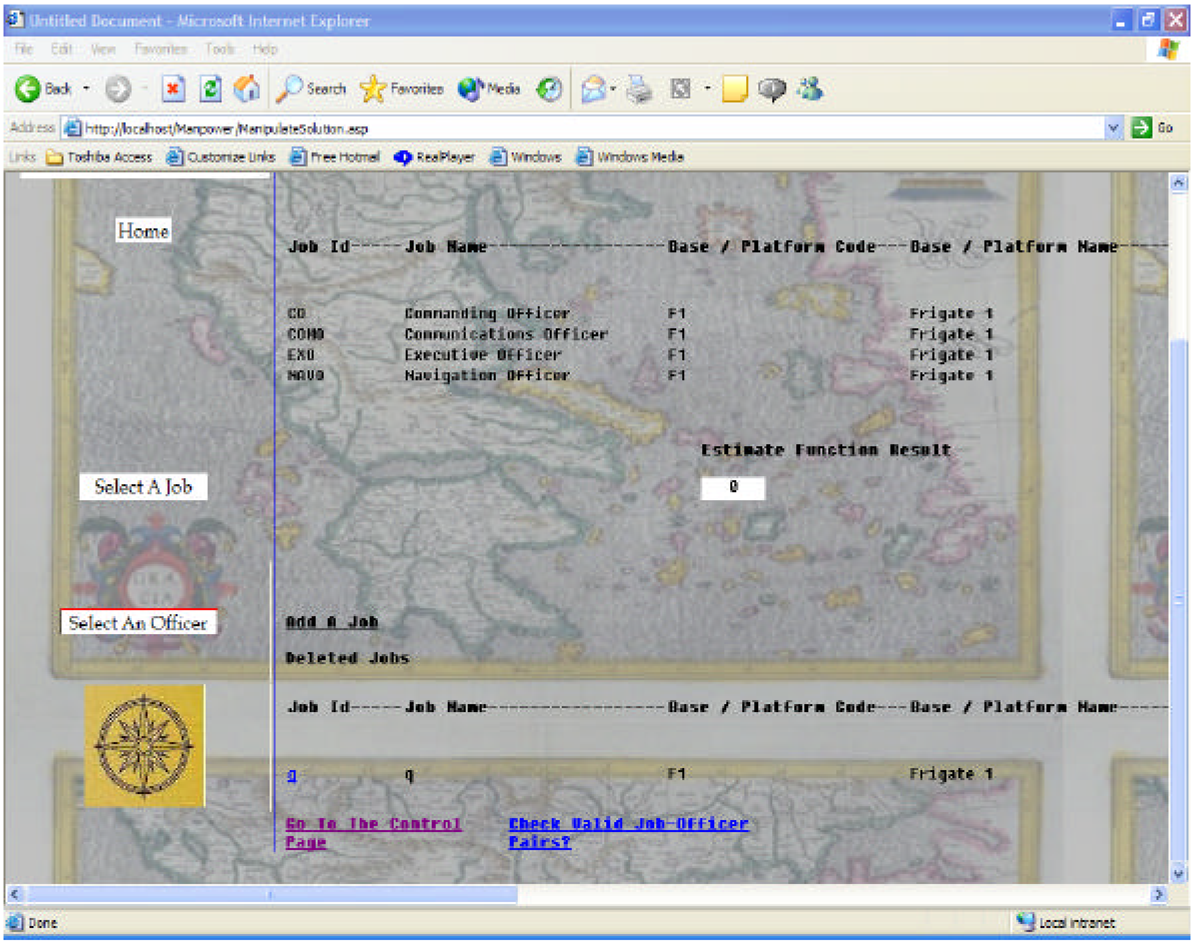


Figure 87. The Page the Detailer Can Change the Solution (Screen 1)-Manpower Website

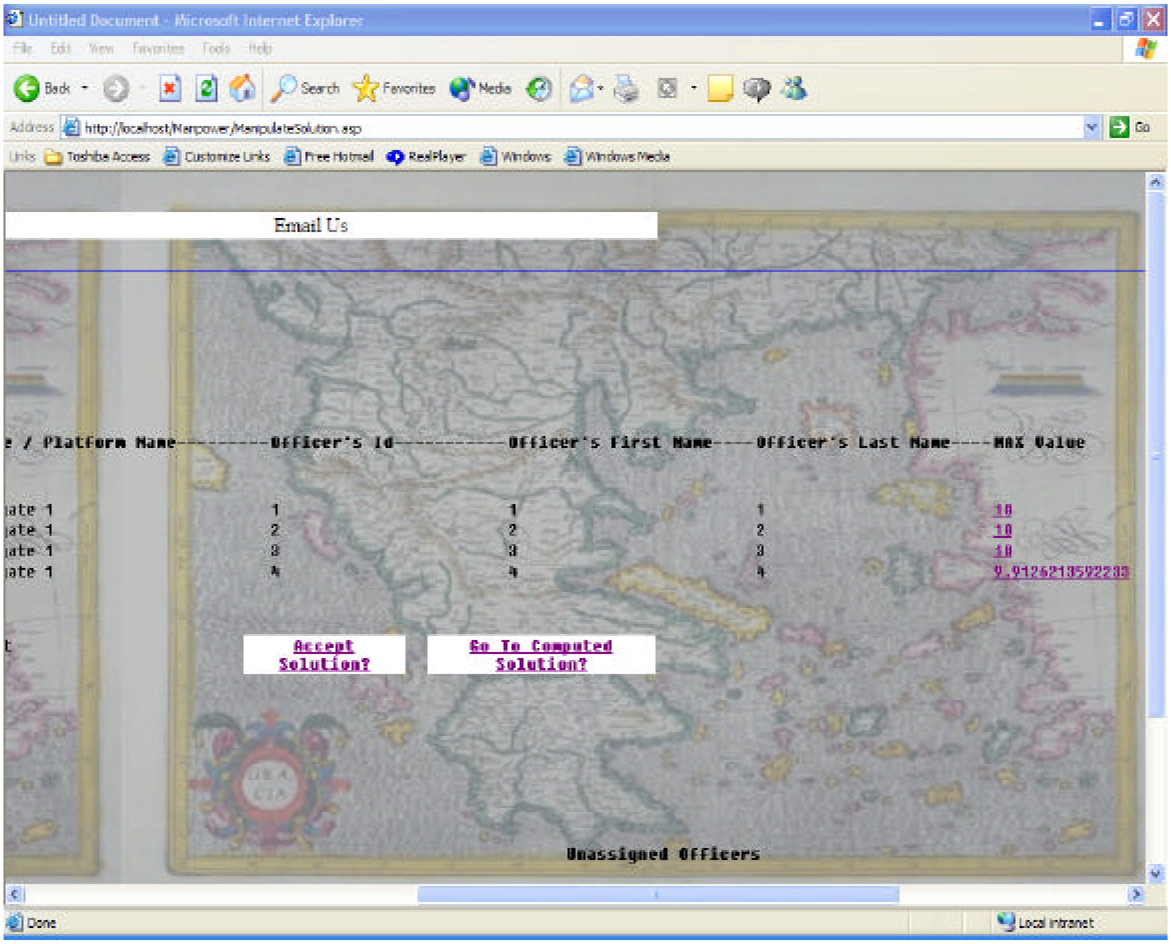


Figure 88. The Page on Which the Detailer Can Change the Solution (Screen 2). On That Page the Detailer Selects the MAX Value 10 Link That Corresponds to Job Commanding Officer and Officer 1-Manpower Website.

As soon as the detailer selects a specific job, the job and the corresponding officer appear under the Deleted Jobs and Unassigned Officers lists accordingly. At the same time the Estimate Function Result appears which shows how worse the detailers change is compared with the algorithms solution.

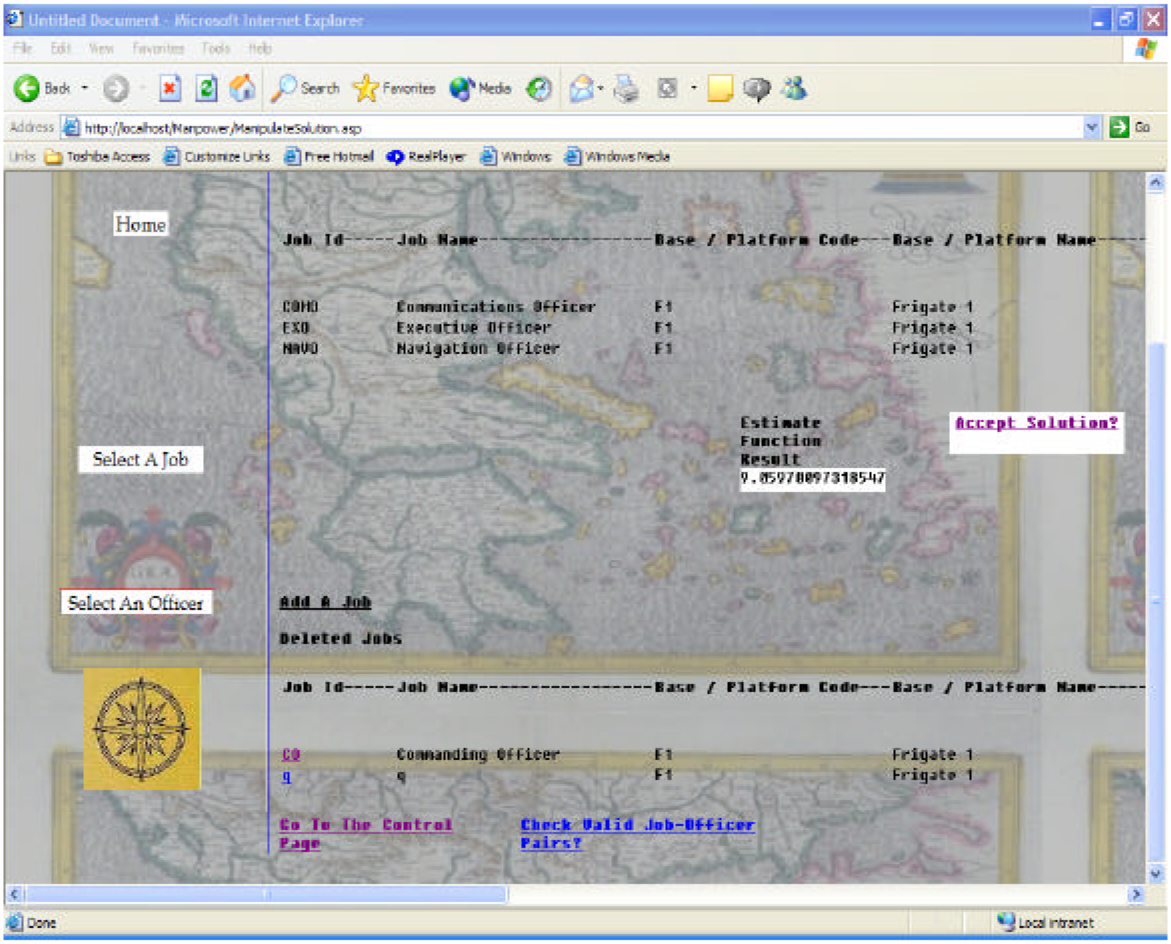


Figure 89. The Job Commanding Officer and Officer 1 is Deleted from the Solution (Screen 1)-Manpower Website.

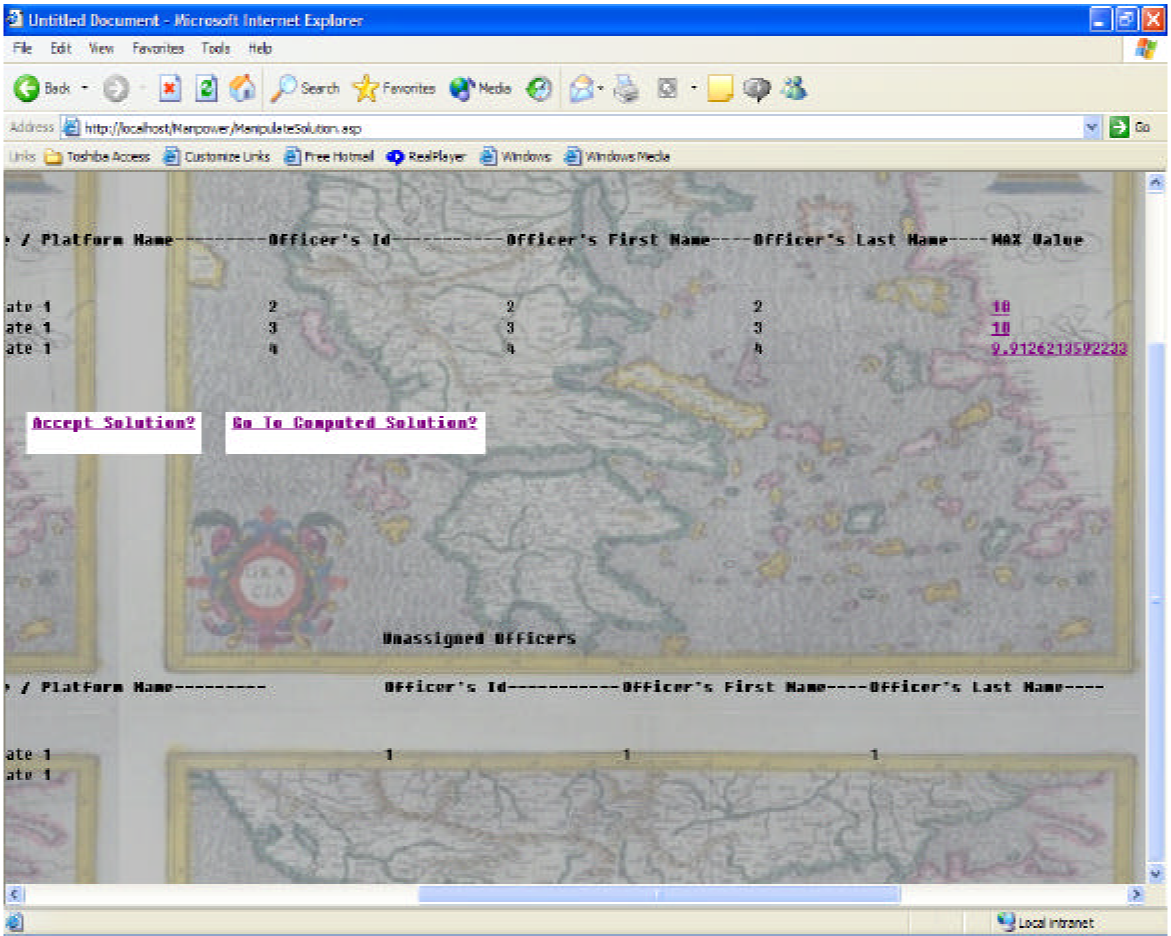


Figure 90. The Job Commanding Officer and Officer 1 is Deleted from the Solution (Screen 2)-Manpower Website.

By performing the same sequence of actions the detailer deletes the job Communications Officer and officer 2. The job Communications Officer and officer 2 appear under the Deleted Jobs and Unassigned Officers lists accordingly. The Estimate Function Result changes again.

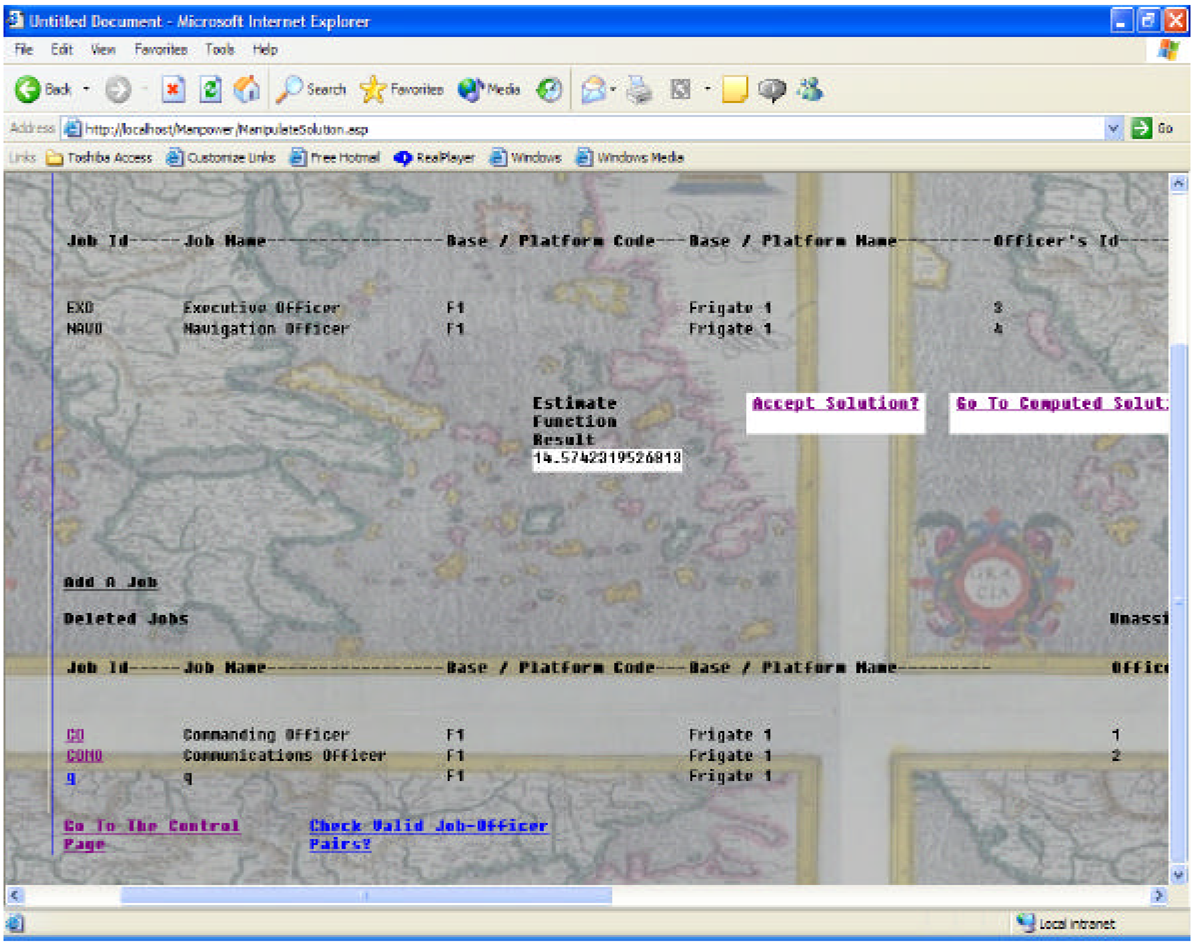


Figure 91. The Job Communications Officer and Officer 2 is Deleted from the Solution (Screen 1)-Manpower Website.

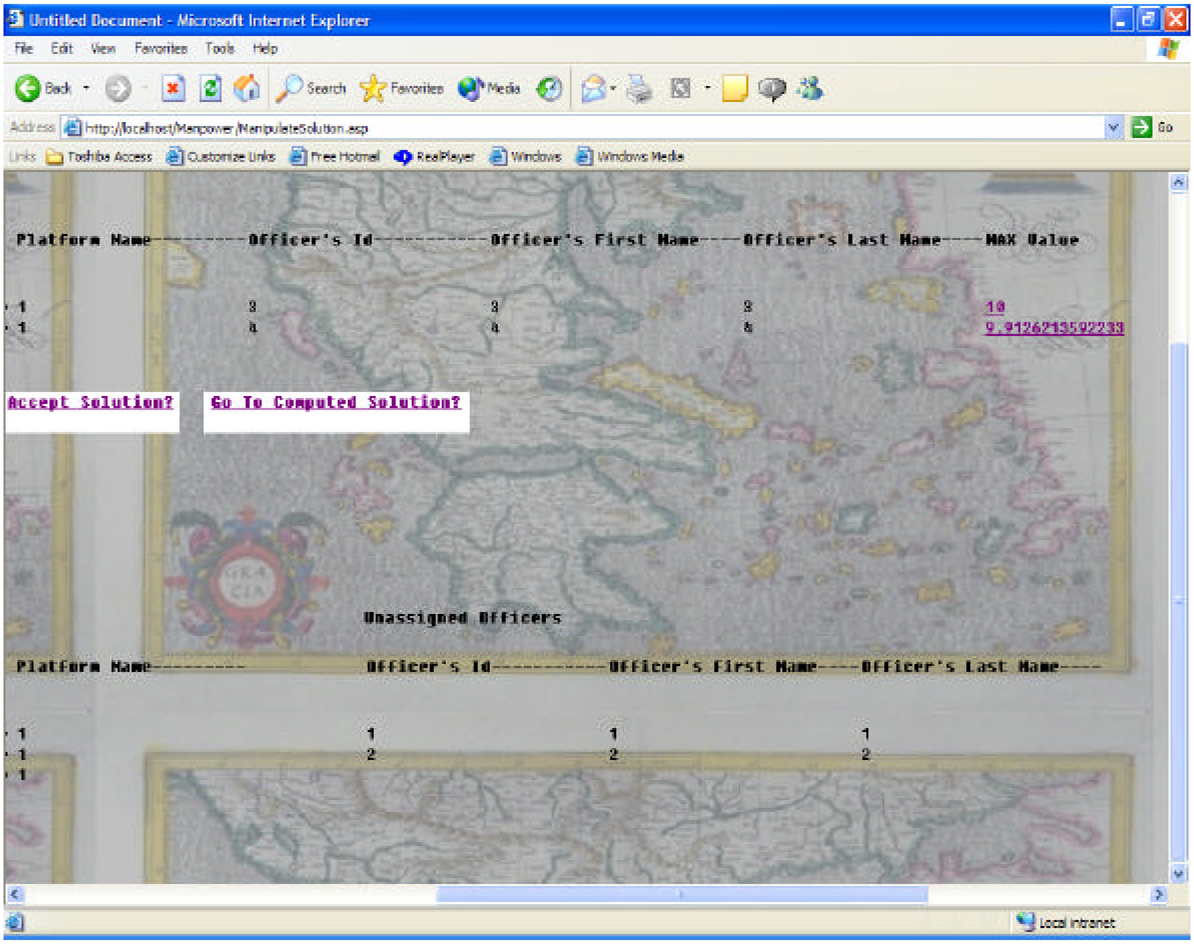


Figure 92. The Job Communications Officer and Officer 2 Is Deleted from the Solution (Screen 2)-Manpower Website.

The detailer then assigns the job Commanding Officer to officer 2 and the job Communications Officer to officer 1. The detailer selects first the job and then the officer that the detailer would like to be assigned to that specific job.

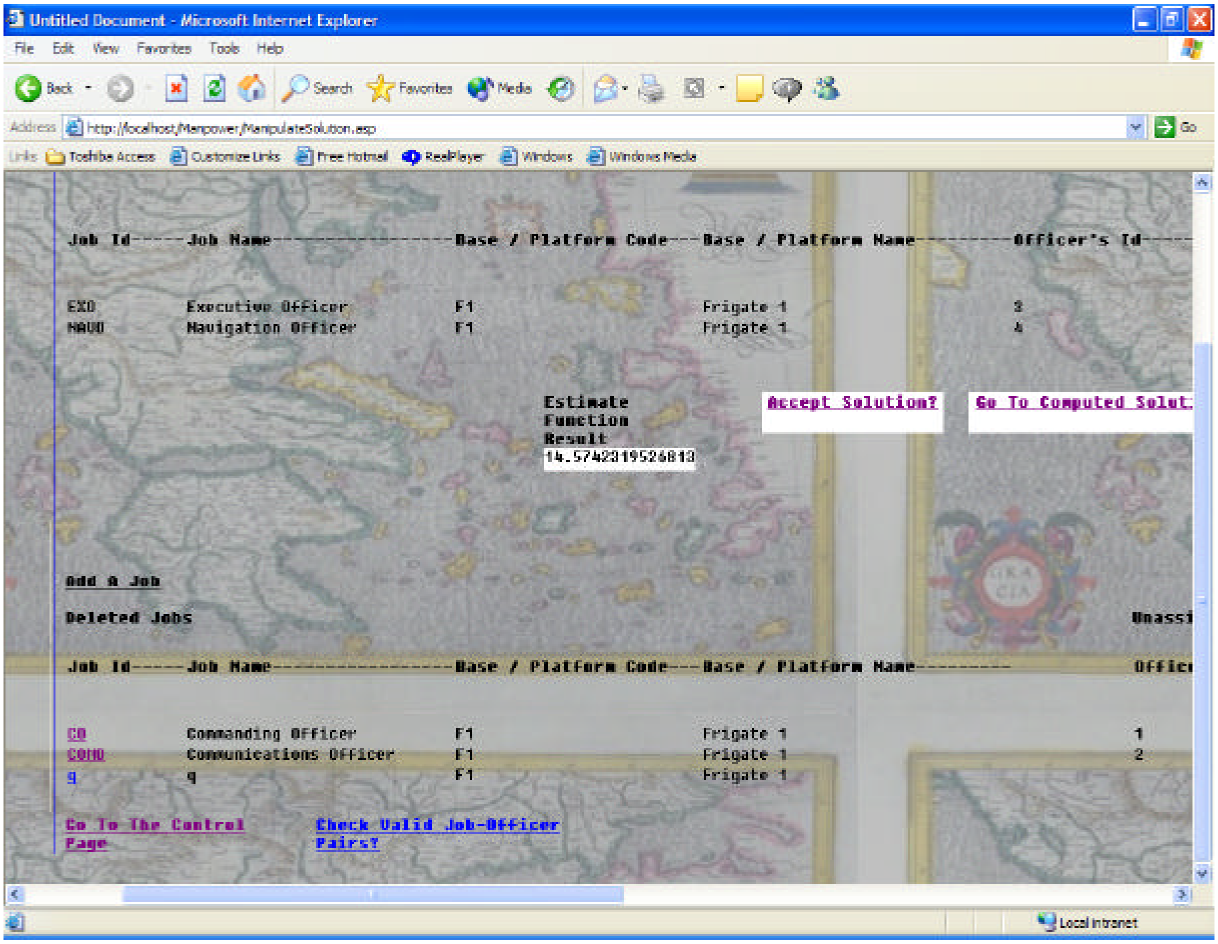


Figure 93. The Detailer Selects the CO Link Under the Deleted Jobs-Manpower Website.

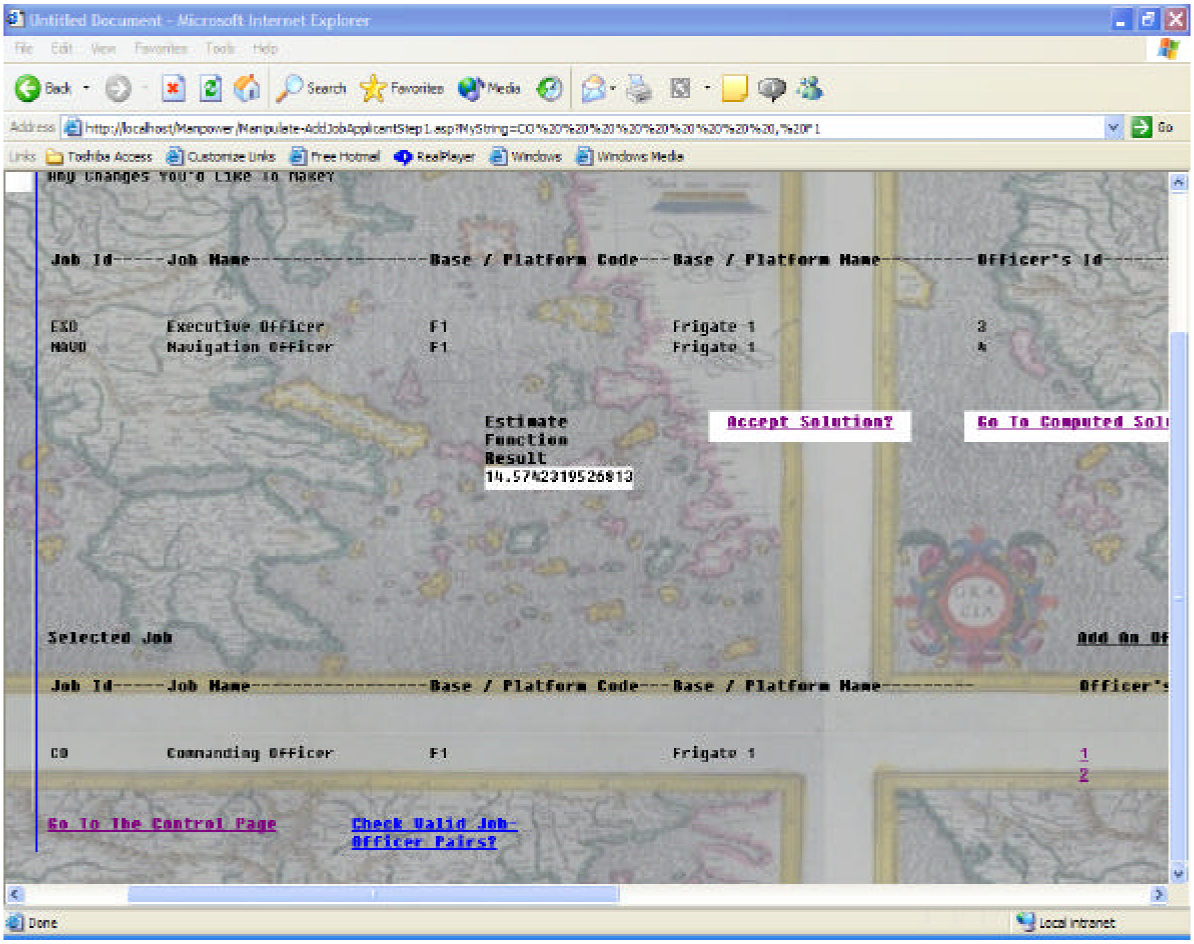


Figure 94. The CO Link is Selected Under 'Selected Job' (Screen 1)-Manpower Website.

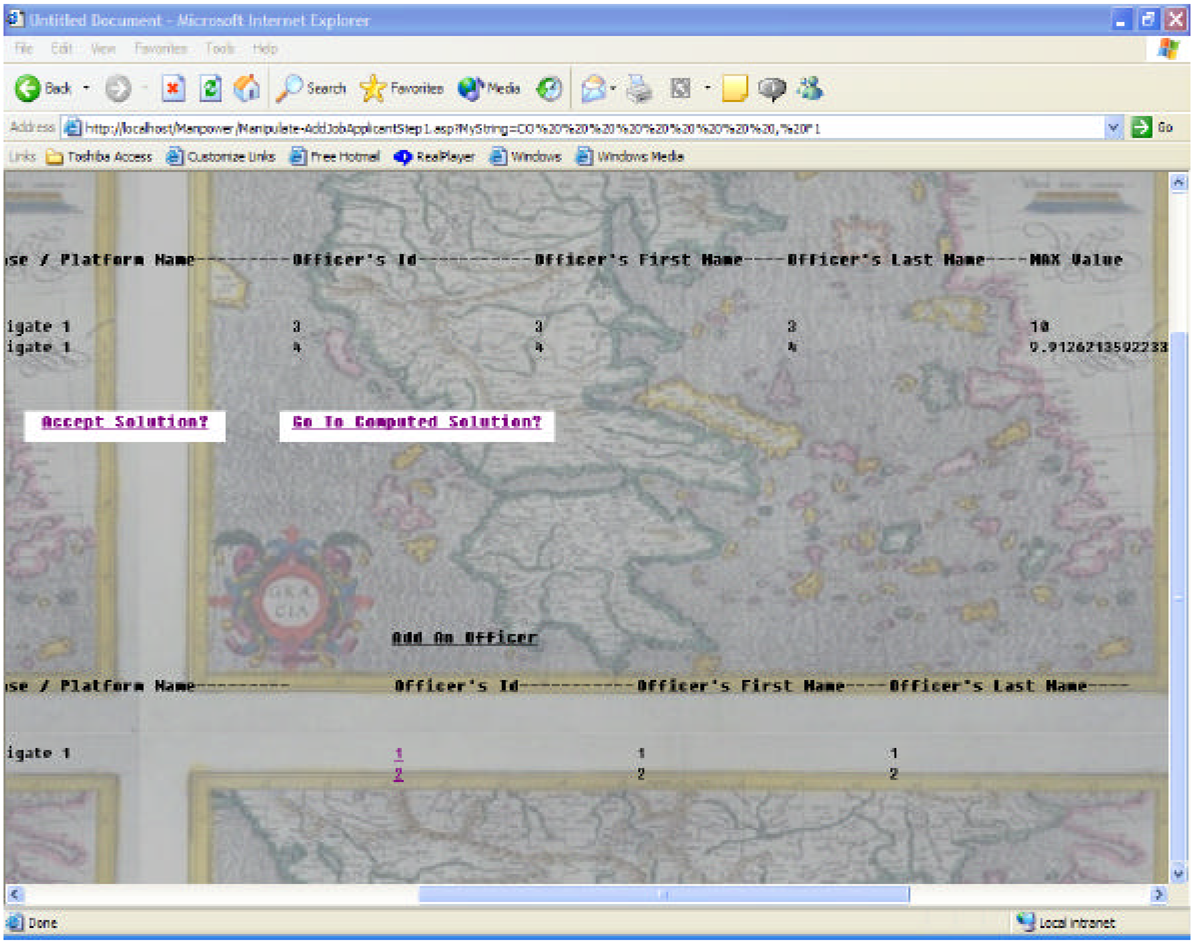


Figure 95. The CO Link Is Selected Under 'Selected Job'. Notice the Available Officers Under 'Add An Officer' (screen 2)-Manpower Website.

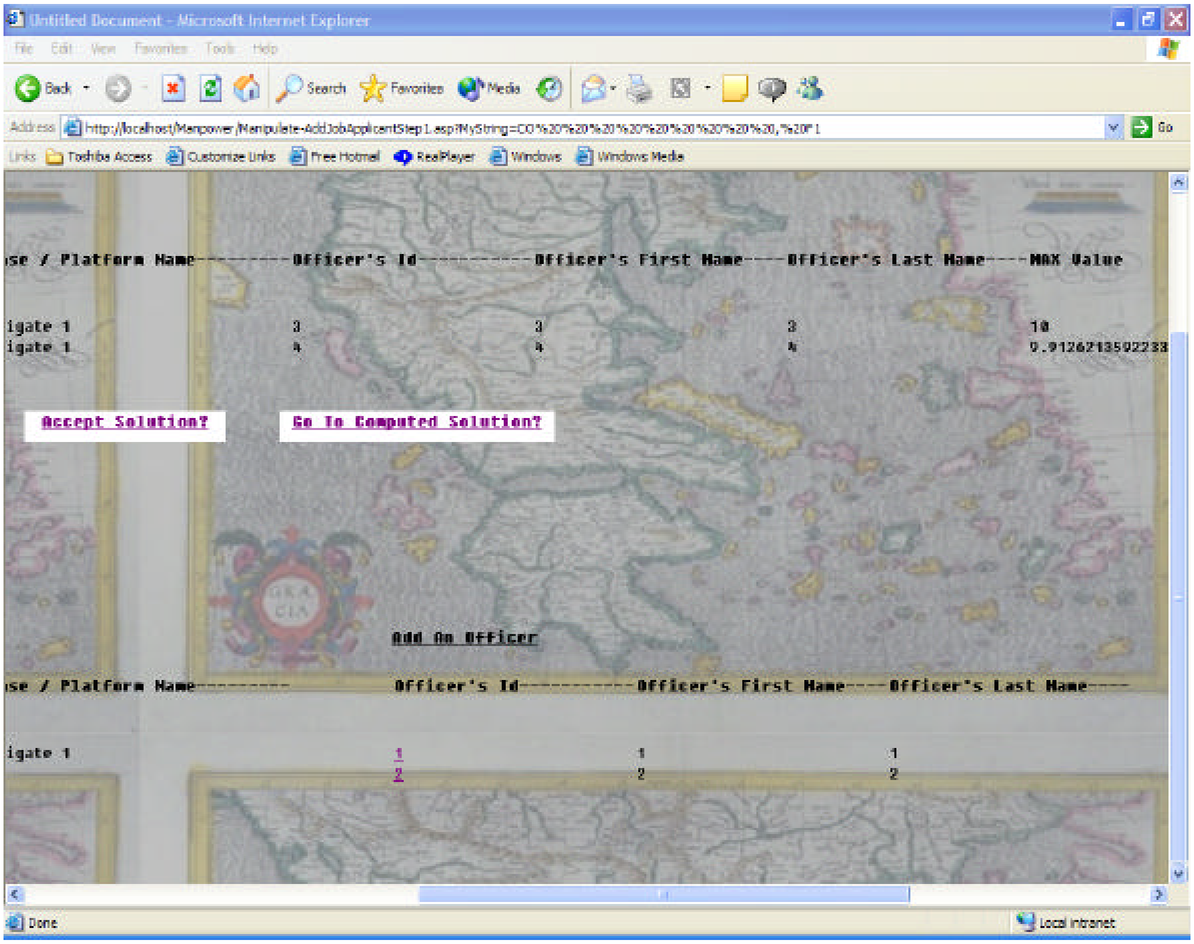


Figure 96. The Detailer Selects Officer 2 Under the ‘Add An Officer’ (Screen 2)-Manpower Website.

By performing all these changes, the Estimate Function Result changes accordingly, so that the detailer can estimate the ‘value’ of his/her changes.

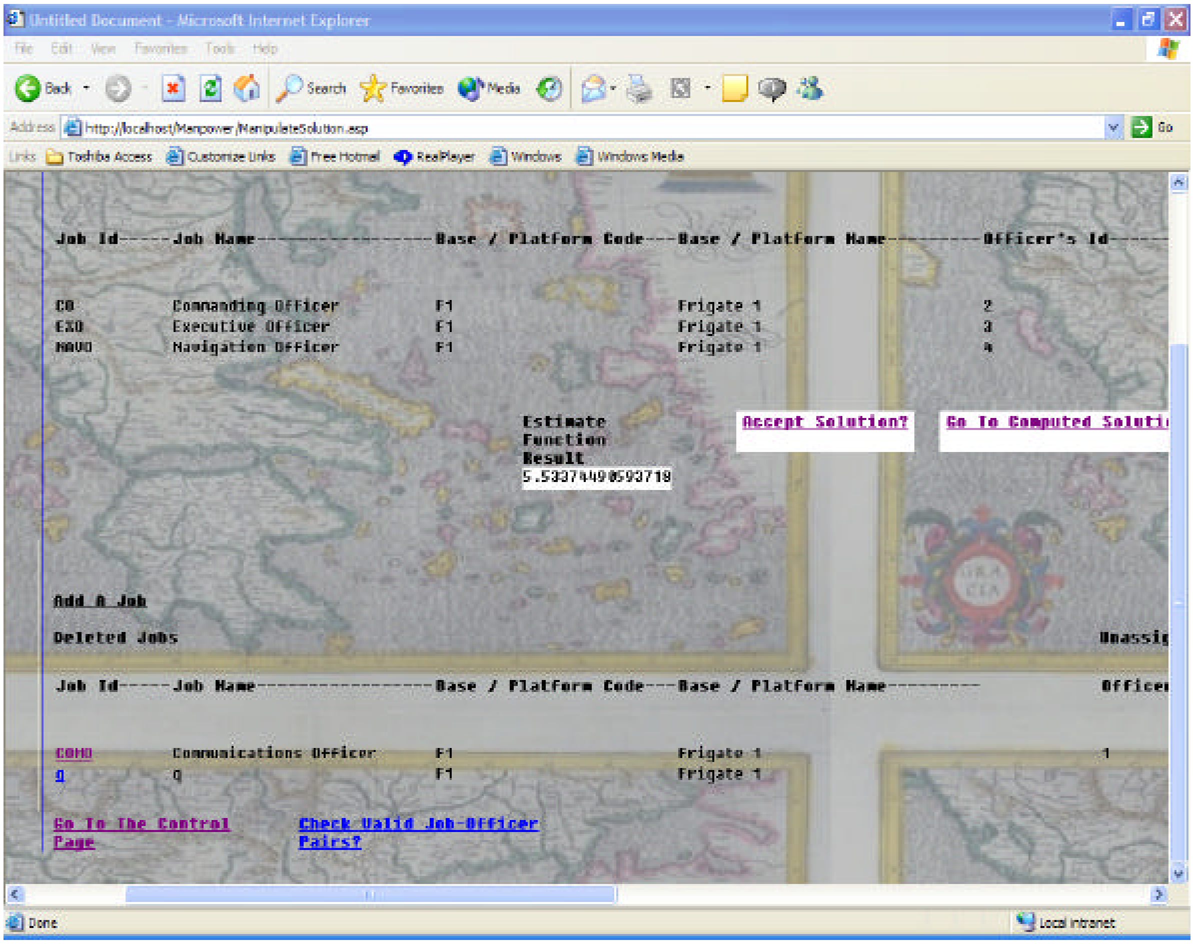


Figure 97. Officer 2 Is Selected. The Job Commanding Officer and Officer 2 Appear in the Solution Domain (Screen 1)-Manpower Website.

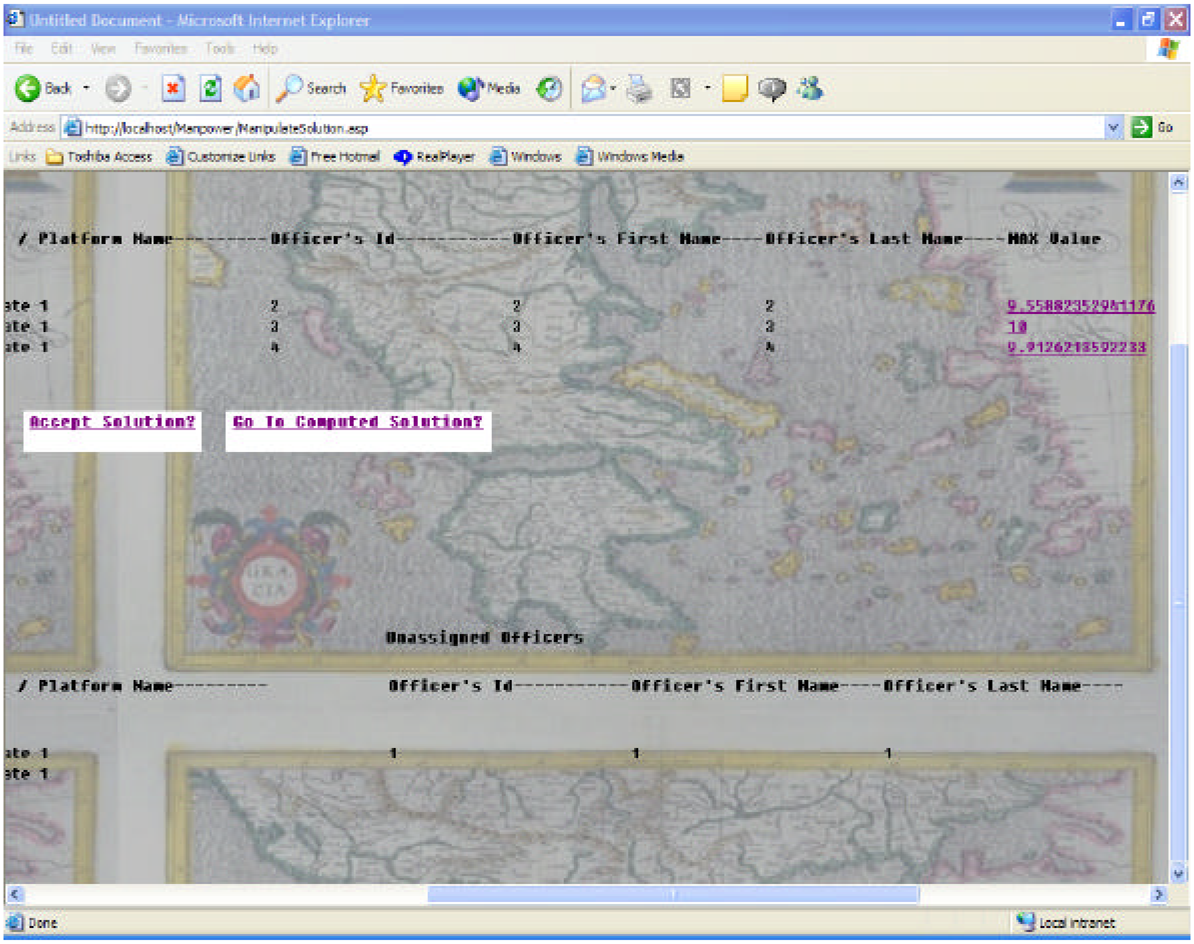


Figure 98. Officer 2 is Selected. The Job Commanding Officer and Officer 2 Appear in the Solution Domain (Screen 2)-Manpower Website.

Following the same sequence of actions, the job Communications Officer and officer 1 are selected. They both appear in the solution domain.

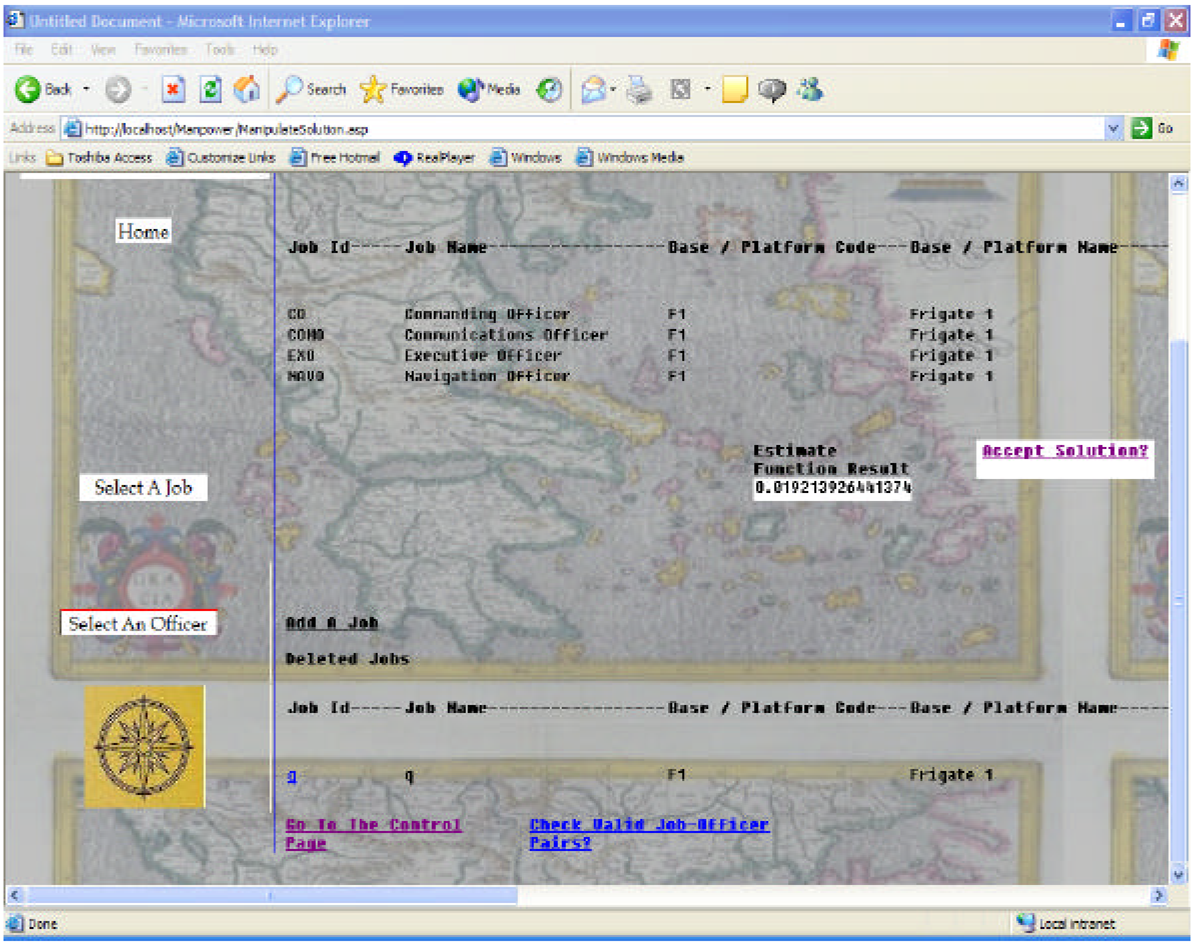


Figure 99. Job Communications Officer and Officer 1 Are Selected (Screen 1)-Manpower Website.

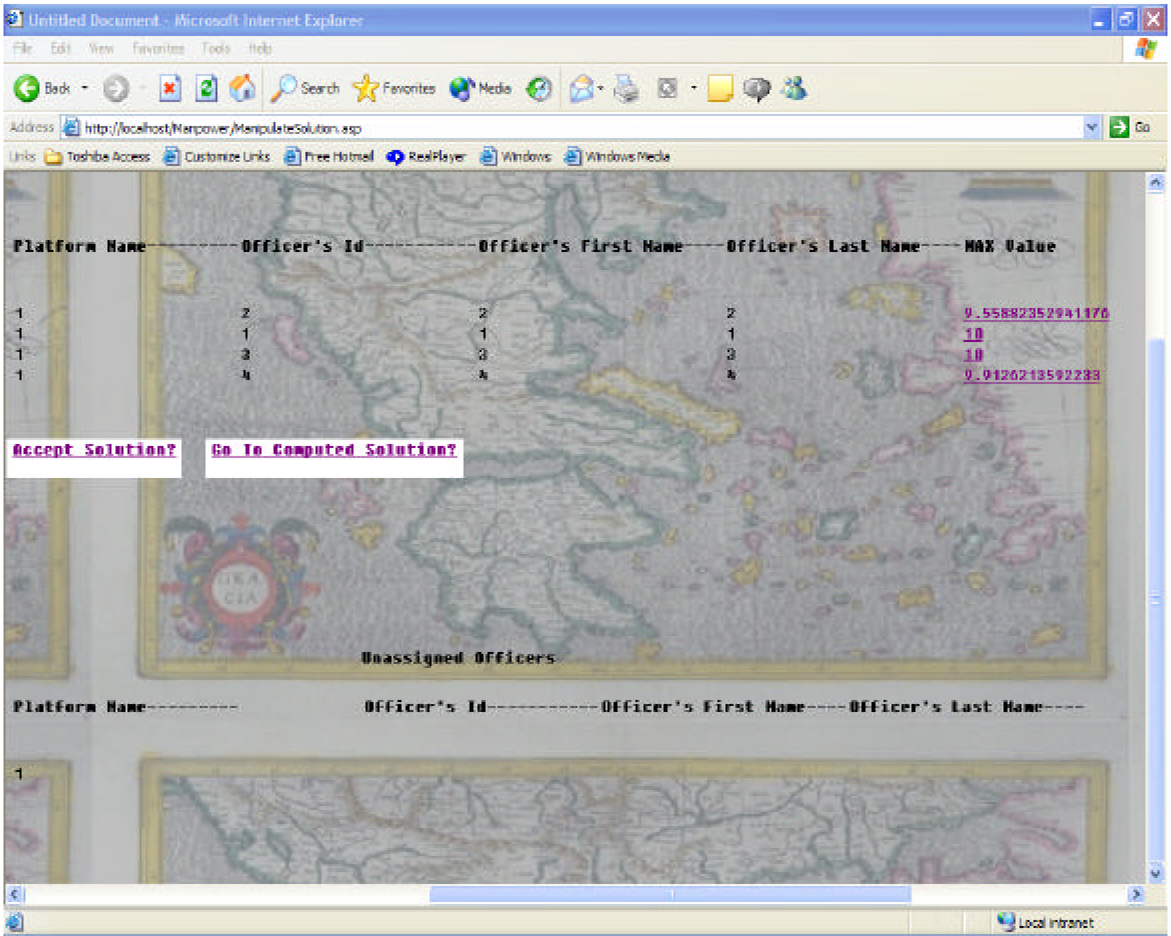


Figure 100. Job Communications Officer and Officer 1 Are Selected (Screen 2)- Manpower Website.

As soon as the detailer has made up his mind, he/she can accept the solution by selecting the ‘Accept Solution’ link. The detailer can also return to the computed solution by selecting the ‘Go To Computed Solution’ link and then accept the solution.

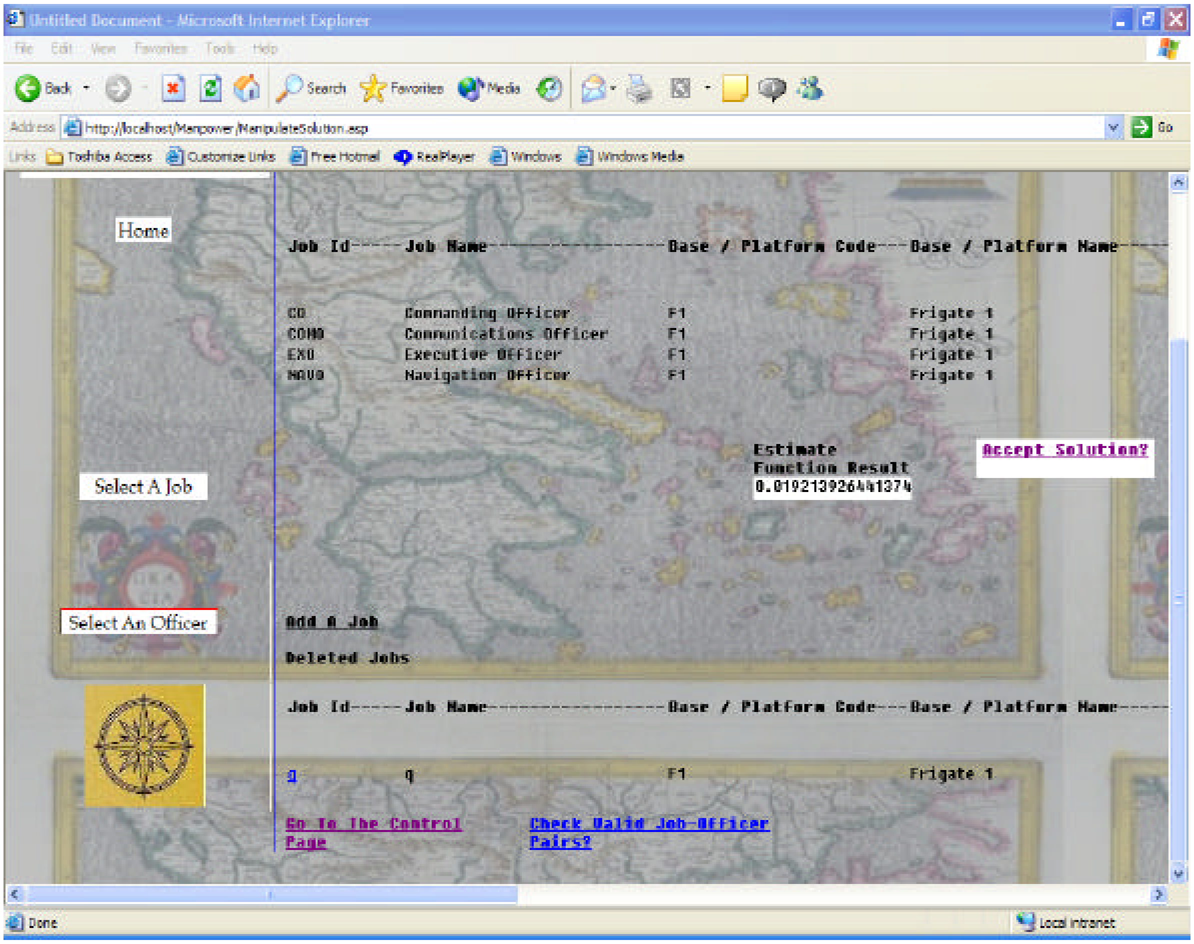


Figure 101. The Detailer Accepts the Solution. The 'Accept Solution' Link is Selected-Manpower Website.

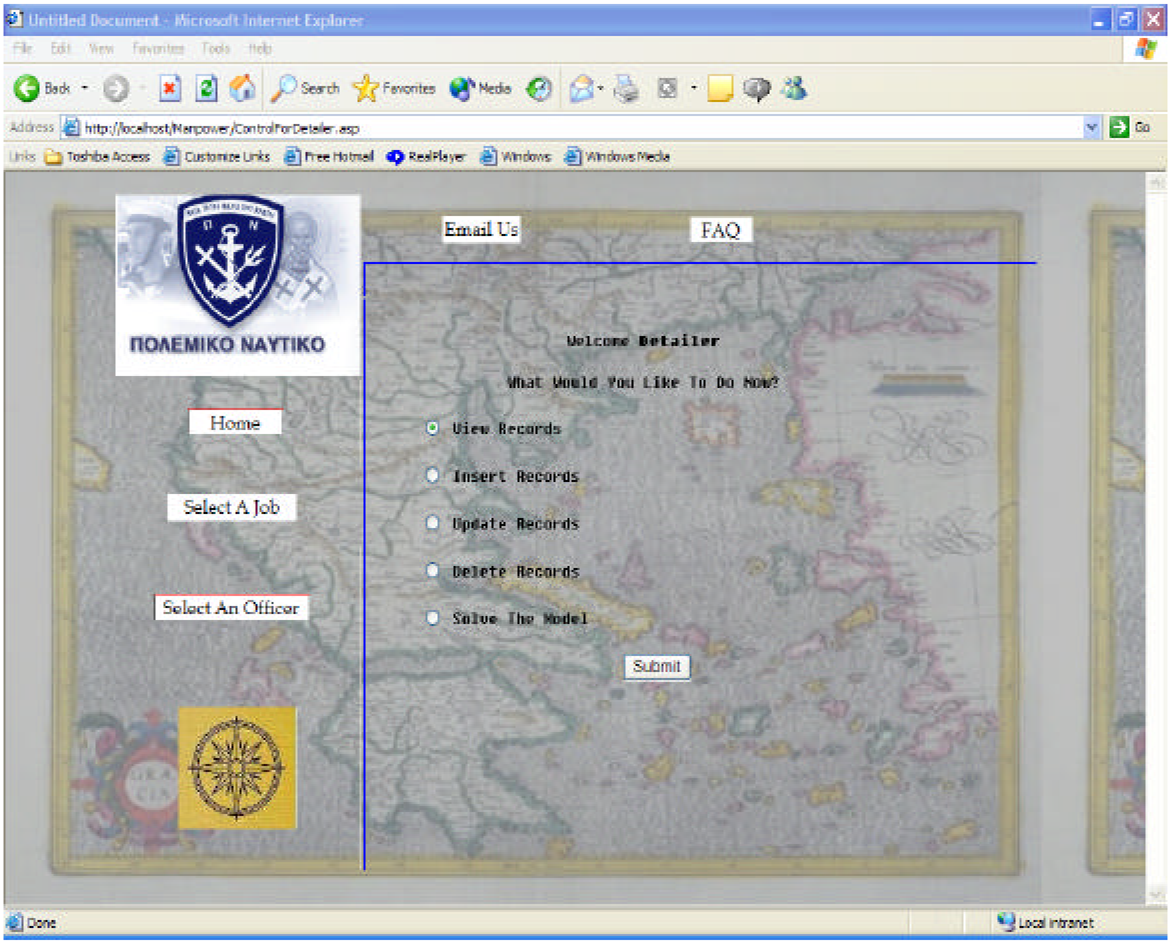


Figure 102. The Solution Is Accepted. The Detailer Goes Back to the Detailer Control Page-Manpower Website.

E. SYSTEM ARCHITECTURE

In this section a description about Microsoft SQL Server, Microsoft IIS 5.0 architecture is provided alongside with some features of the Windows XP Professional NTFS operating system, under the perspective of the Manpower Database and Website needs.

1. Microsoft SQL Server 2000-Management

Microsoft SQL Server 2000 provides many desirable features for the Manpower Database:

a. Database Management

The figure below shows the SQL Server Enterprise Manager. It provides an easy-to-use interface that enables the manager to perform any desired tasks by using menus and dialog boxes rather than complex command line instructions.

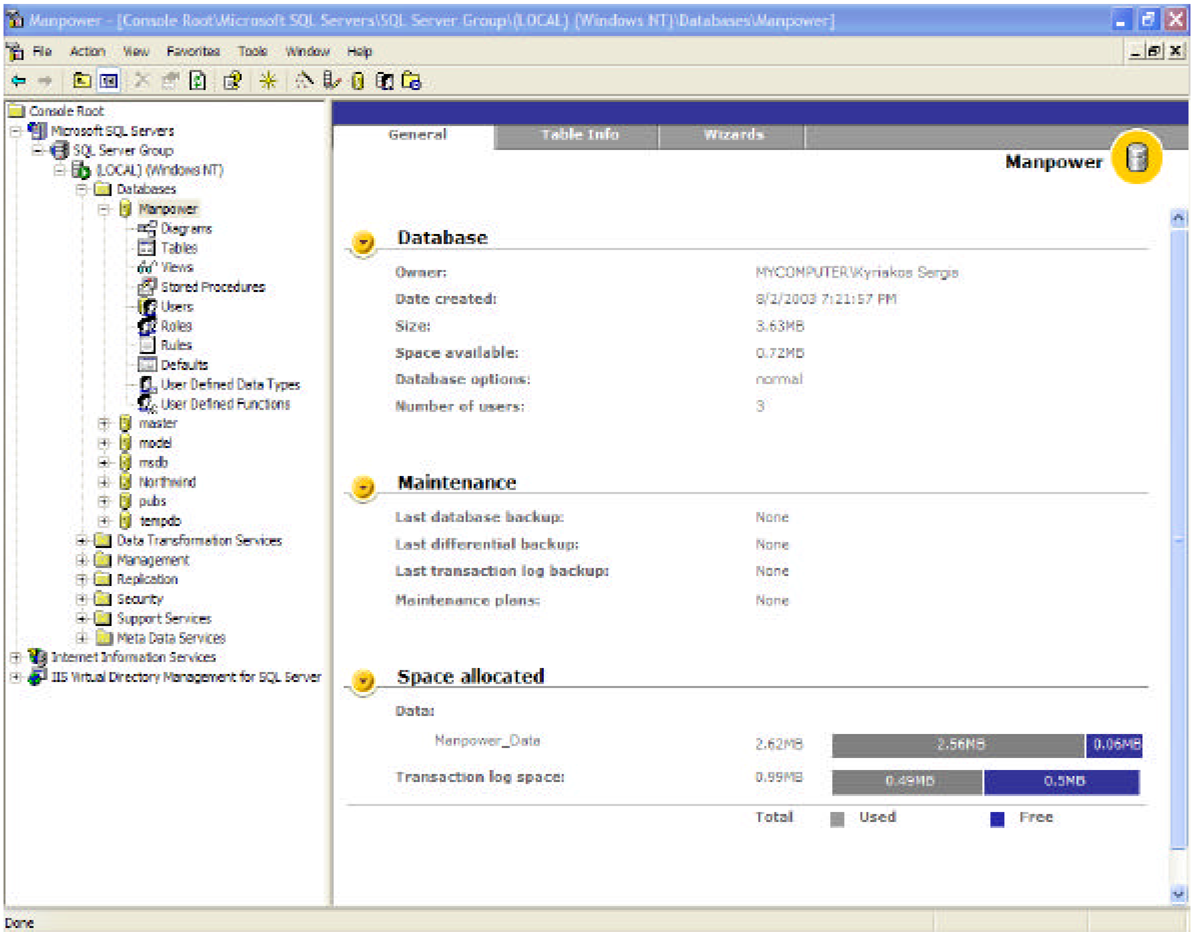


Figure 103. Microsoft SQL Server 2000 Enterprise Manager-Manpower Database.

b. Stored Procedures

Stored Procedures are predefined queries whose values are variables that are not defined until run time. Stored procedures can be nested up to 32 levels deep. In the Figure below, we see an example of the UpdatePhoneData stored procedure used in the Manpower database. This procedure receives the ApplicantId, HomePhoneNumber, CellPhoneNumber and OtherPhoneNumber values from the web server, performs the UPDATE query based on these values and updates the PHONE table. The sign @ characterizes a parameter as a variable and is put in front of that parameter.

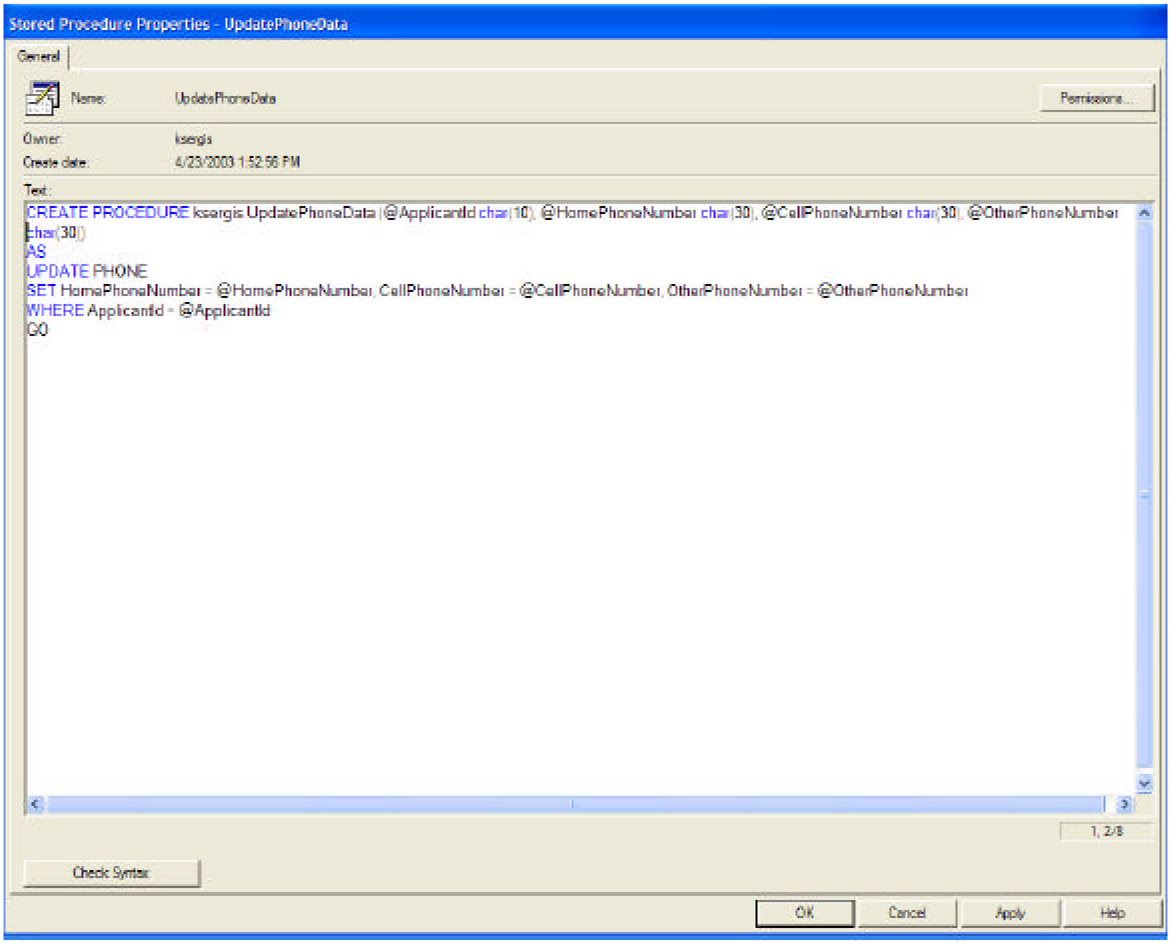


Figure 104. Use of Stored Procedure-Manpower Database.

Moreover, Stored Procedures use a special script language, Transact-SQL, which helps the manager to create code in order to perform administrative tasks.

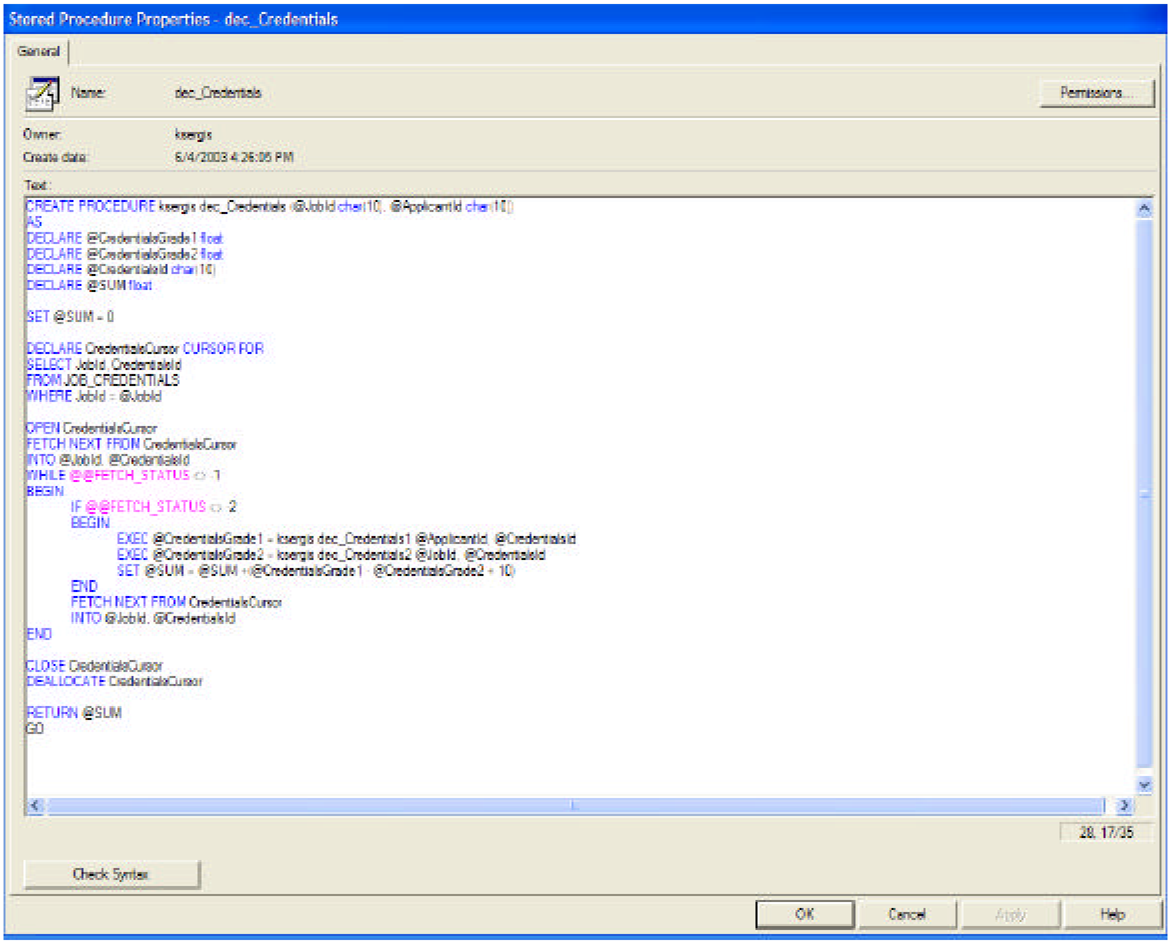


Figure 105. Transact-SQL Code Example-Manpower Database.

c. Database Diagrams

SQL 2000 Server provides an easy to use interface for viewing the structure of the database and creating relationships among tables. Relationships can be created by dragging and dropping primary keys from one table to the foreign key reference in another table. For complex databases with hundreds of tables, multiple diagrams with differing configurations can be created.

d. Multiple Ways to Construct Queries

SQL 2000 Server provides also Query Builder Wizards, Query Design Grid similar to Access, and an "English Query" engine for defining queries through English phrases rather than SQL syntax. It provides SQL Query Analyzer, which is a powerful tool that helps the manager check queries or even stored procedures.

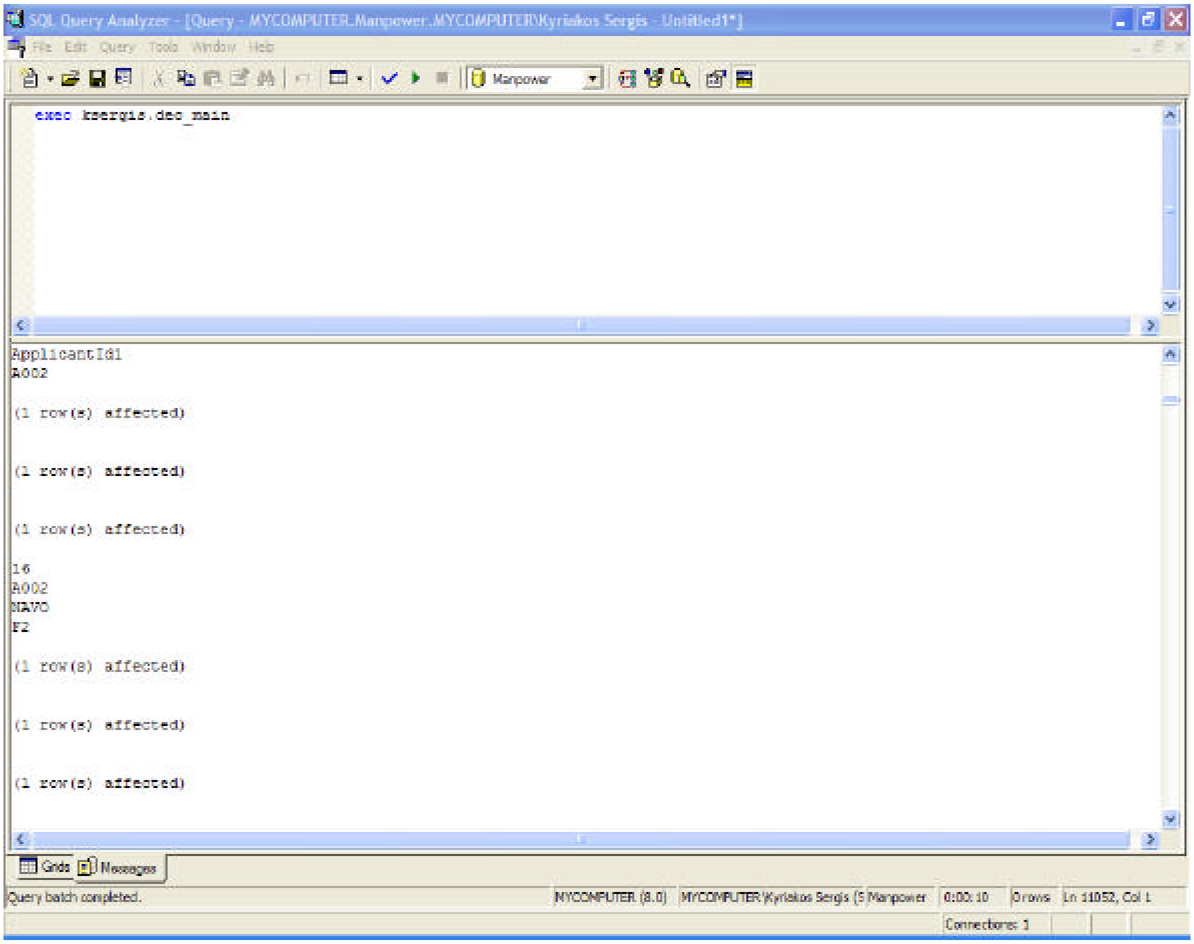


Figure 106. Use of SQL Query Analyzer-Manpower Database.

2. Manpower Database and Website-Security Issues

a. Security Modes-Manpower Database

SQL Server 2000 has two security modes. The first one is Windows Authentication Mode and the second one is Mixed Mode. In the first mode, a user needs to login on the Windows domain only. He is authenticated automatically as a valid SQL Server 2000 user. In the Mixed mode the user has to be authenticated to both the Windows domain and the SQL Server 2000. The Mixed mode is more secure and allows the users to work from different OS (Mac, Novell etc.), while the Windows Authentication mode does not require the user to have multiple passwords. In the Manpower database the mixed mode is selected for the reasons mentioned above.

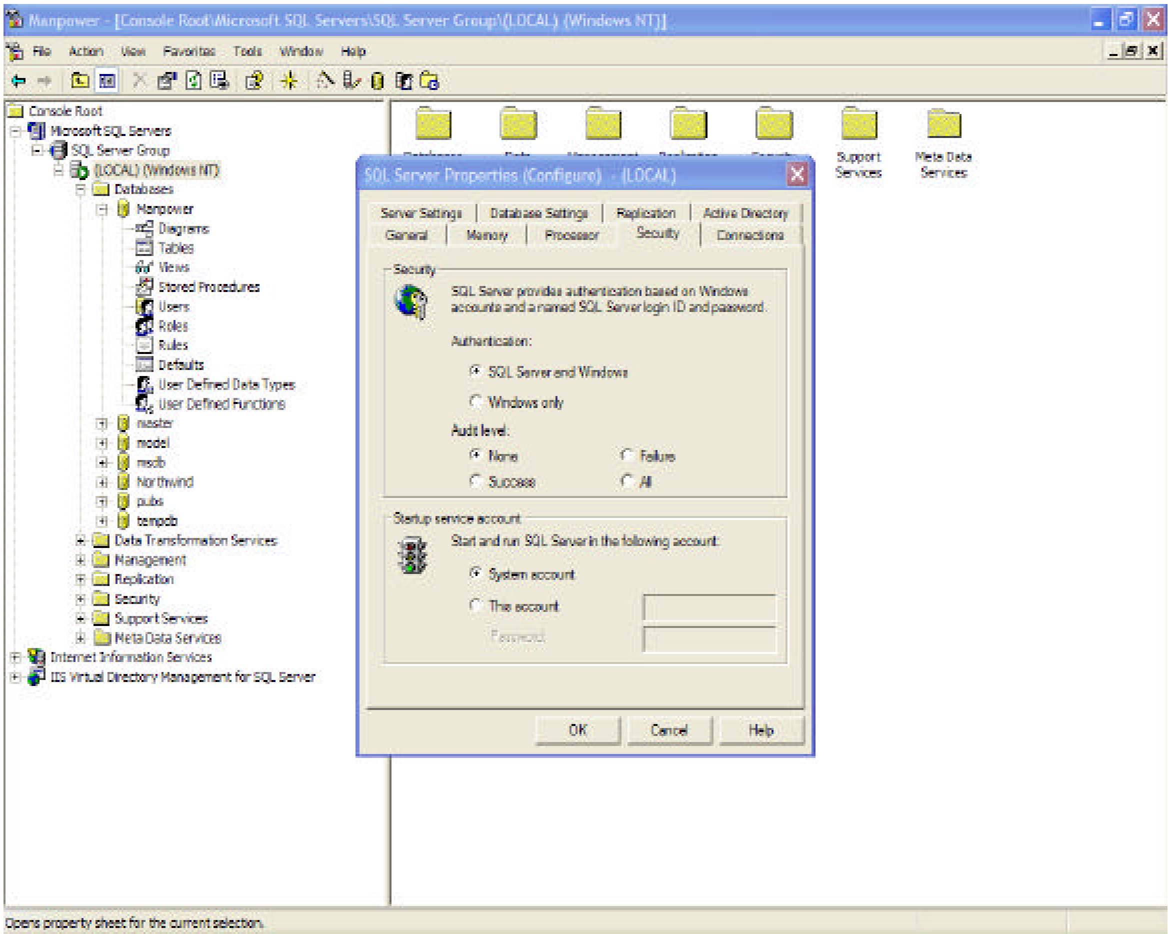


Figure 107. SQL Server 2000 Authentication Mode-Manpower Database.

b. Logins-Manpower Database

A SQL Server 2000 login, gives the server users access to SQL Server as a whole but not to the resources, like the Manpower database, inside. A Standard Login is necessary for the mixed security mode, since Mac or Novell clients need to be authenticated independently of the windows domain. A Standard Login is created for the detailer for the Manpower database.

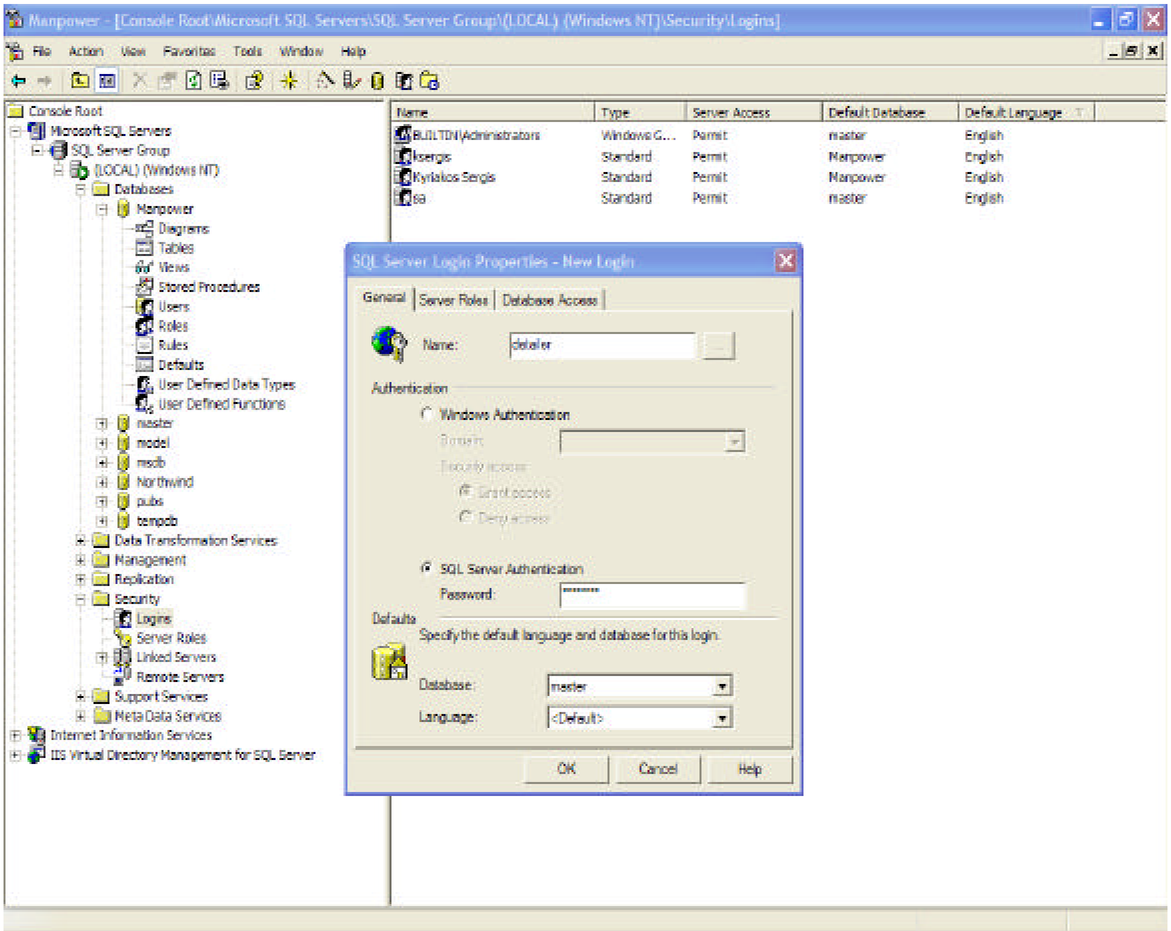


Figure 108. Standard Login-Creation of Detailer Login for the Manpower Database.

c. Manpower Website NTFS Permissions

The Manpower Website files are organized in a manner based on the Manpower Website users, the officer, the command and the detailer. For that purpose three groups are created, the officer group, the command group and the detailer group. Every officer belongs to the officer group, every command belongs to the command group and the detailer to the detailer group.

The officer directory contains all the above groups. The command directory contains the command and detailer group and finally the detailer directory contains only the detailer group. The permissions are Full Control for every group in every directory.

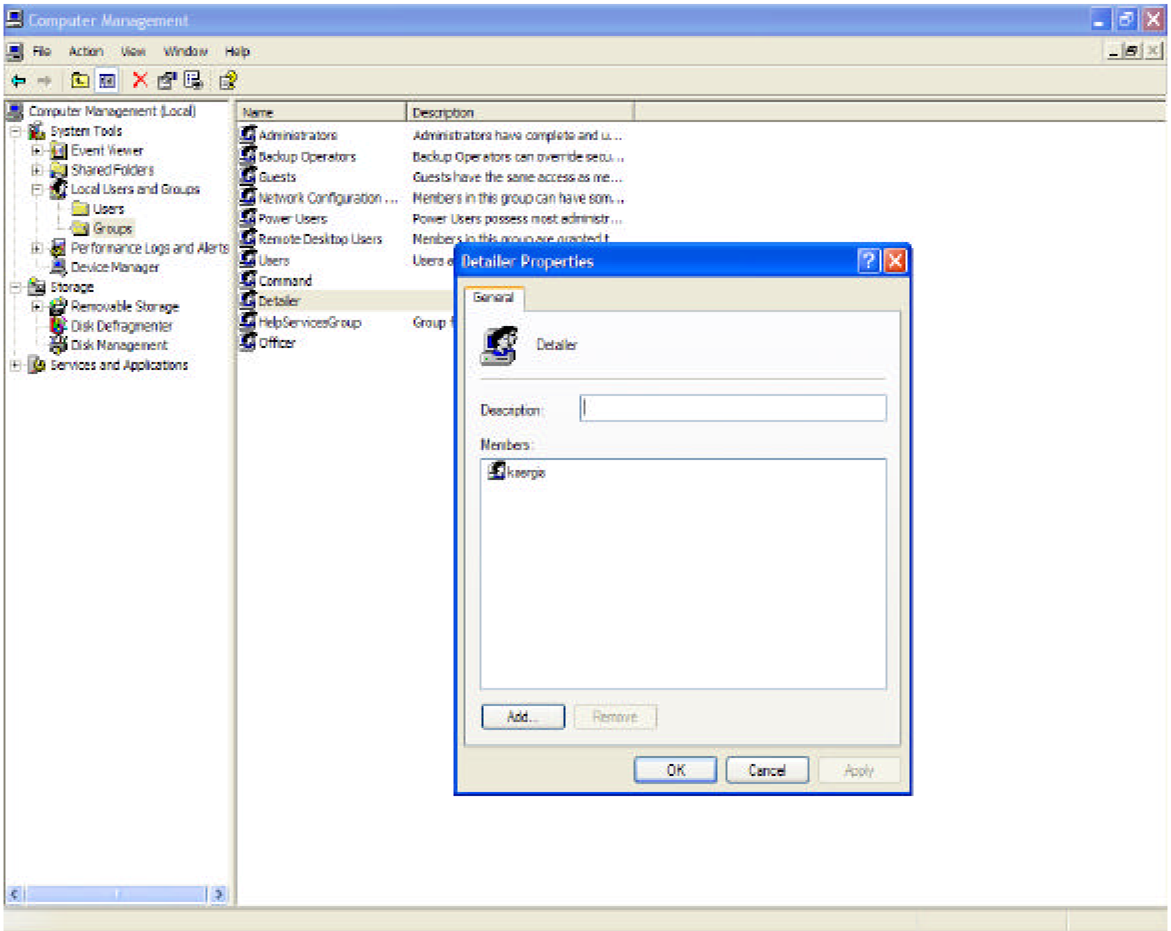


Figure 109. The Detailer 'ksergis' as a Member of the Detailer Group-Manpower Website NTFS Permissions.

d. Manpower Website IIS Permissions

The Manpower Website IIS permissions can be controlled from the Security tab of either the Manpower Website directory or the files belonging to it. The account used for anonymous access can be set to IUSR_MYCOMPUTER or any account of the officer, command or detailer group.

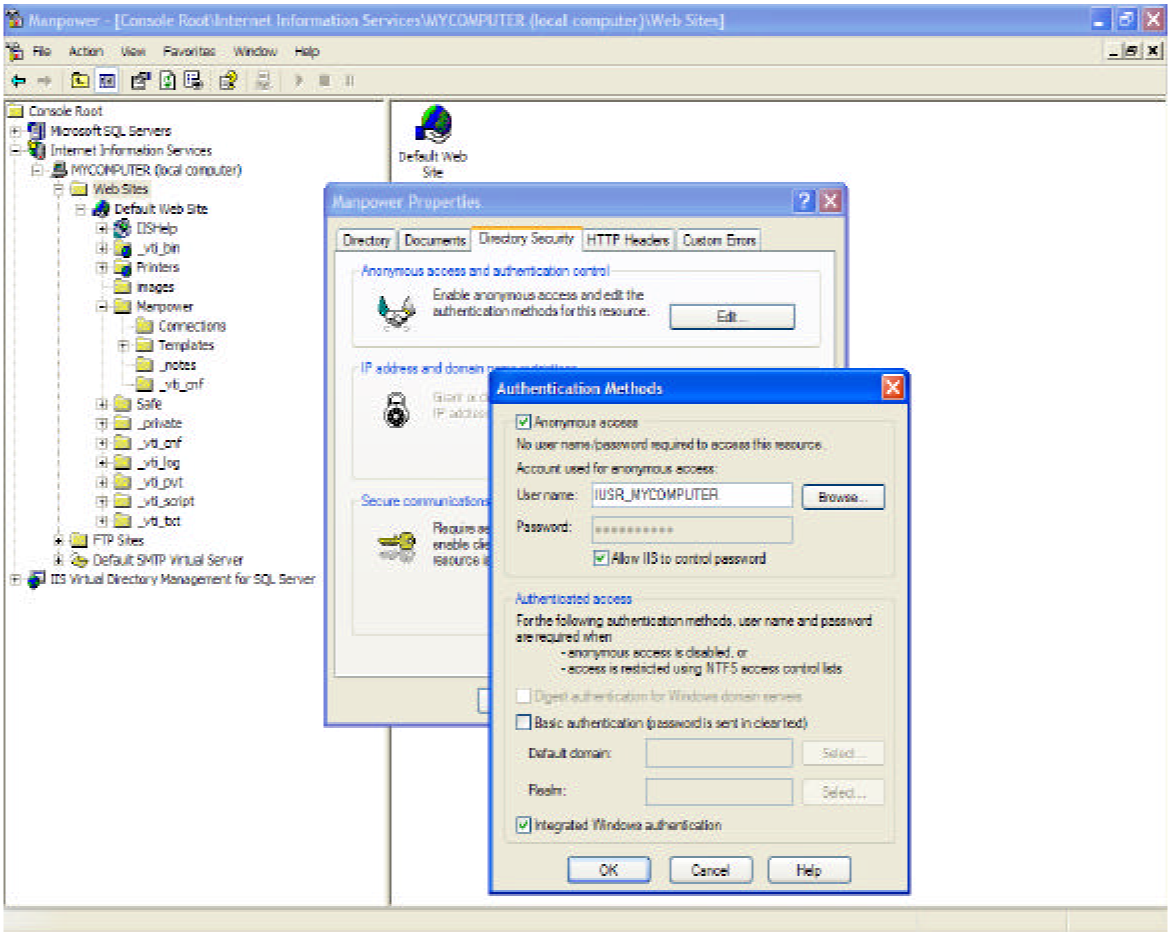


Figure 110. Anonymous Access-Manpower Website IIS Permissions.

e. SQL Server Logs-Manpower Database

SQL Server 2000 provides to the database manager the ability to view current or past logs in order to check any existing delinquencies.

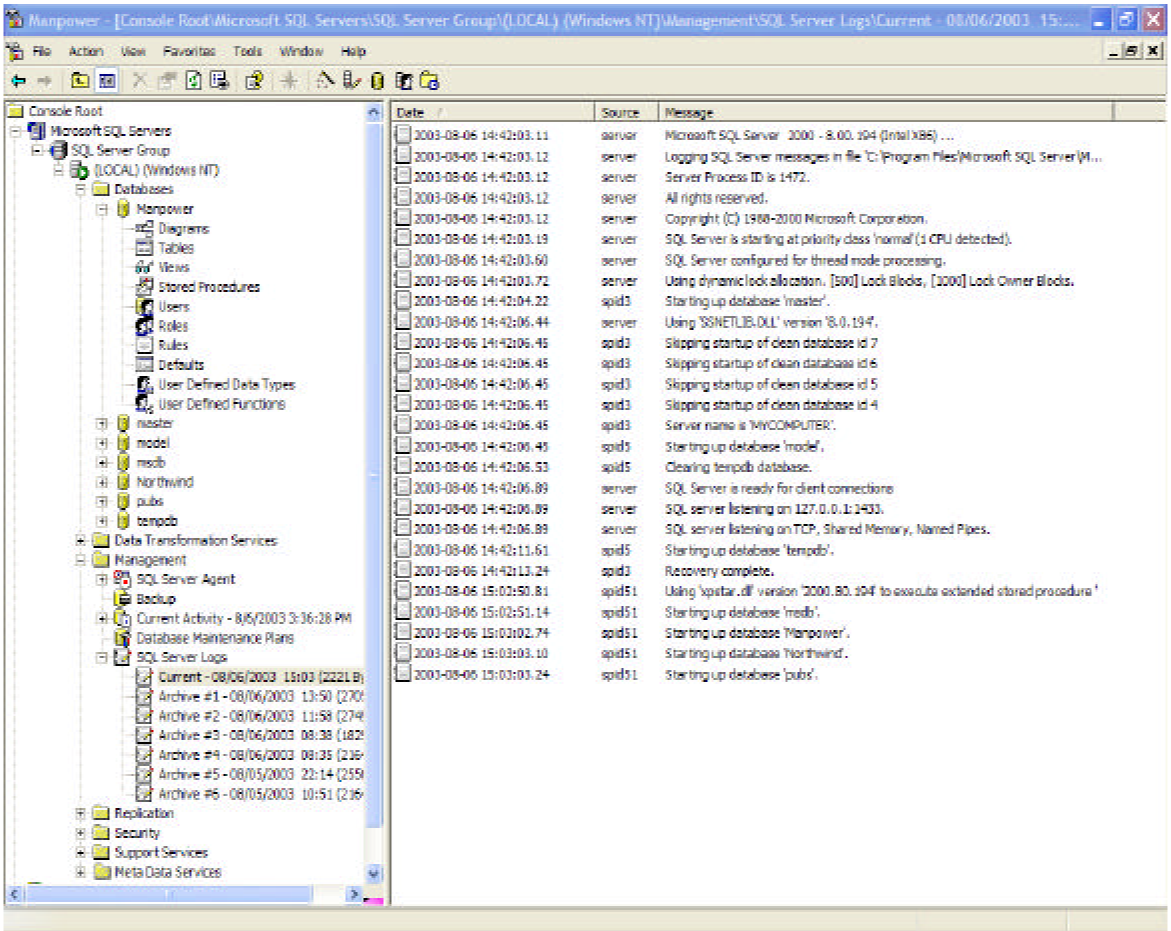


Figure 111. SQL Server Logs-Manpower Database.

3. Microsoft SQL Server 2000-Backup and Maintenance Issues

a. Maintenance Plan

The database manager can arrange maintenance plans to either perform a simple backup, or set up log shipping to a standby server. Below is the first screen shot of performing a maintenance plan.

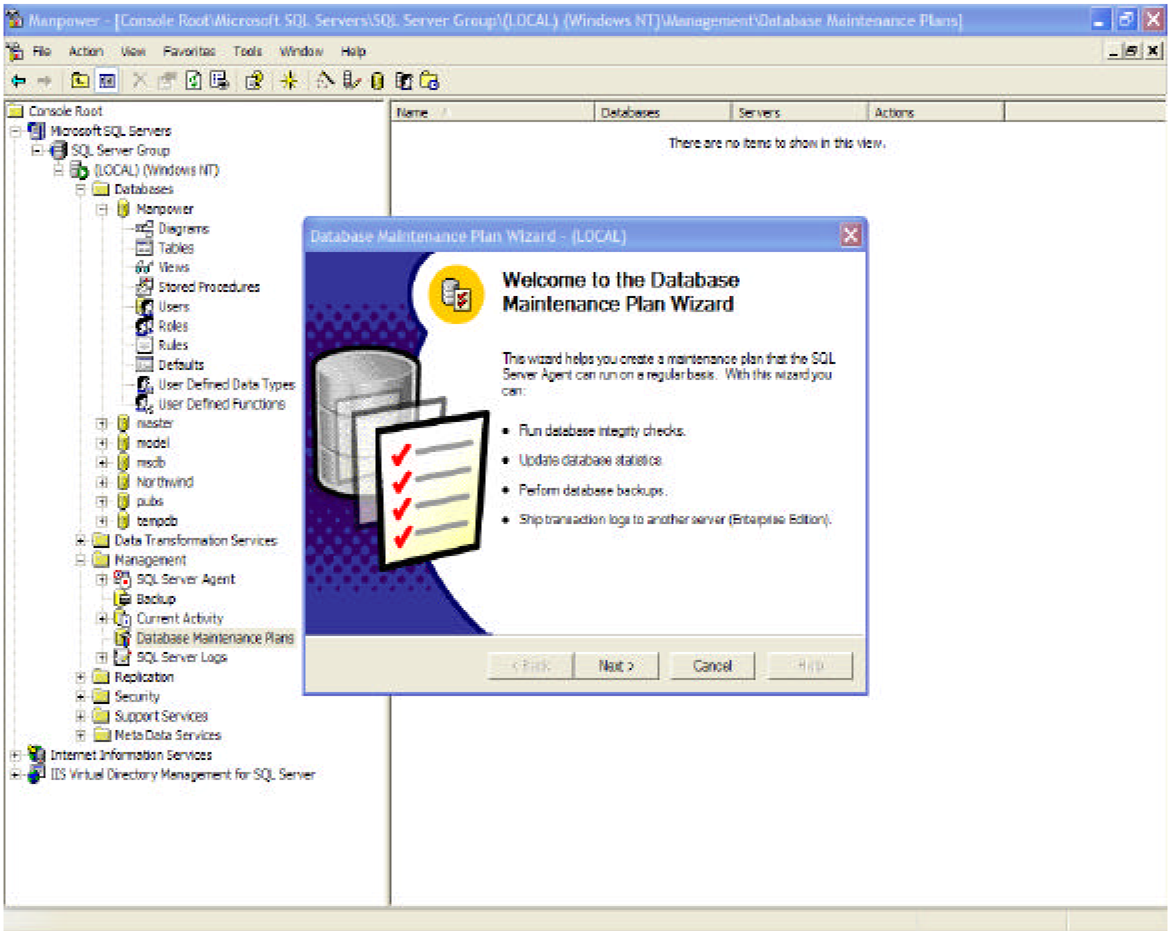


Figure 112. Database Maintenance Plan-Manpower Database.

b. *Backing Up*

The manager has several choices to back up data. The manager can perform a Full backup to back up the entire database, a Transaction log backup to back up the transaction log records, a Differential backup to back up only the data that have changed since the last full backup and finally a Filegroup backup to back up different pieces of the database, based on the various files that make up the database. Since the Manpower database backup mode is Full (instead of Simple), the manager can perform every kind of these four backup choices.

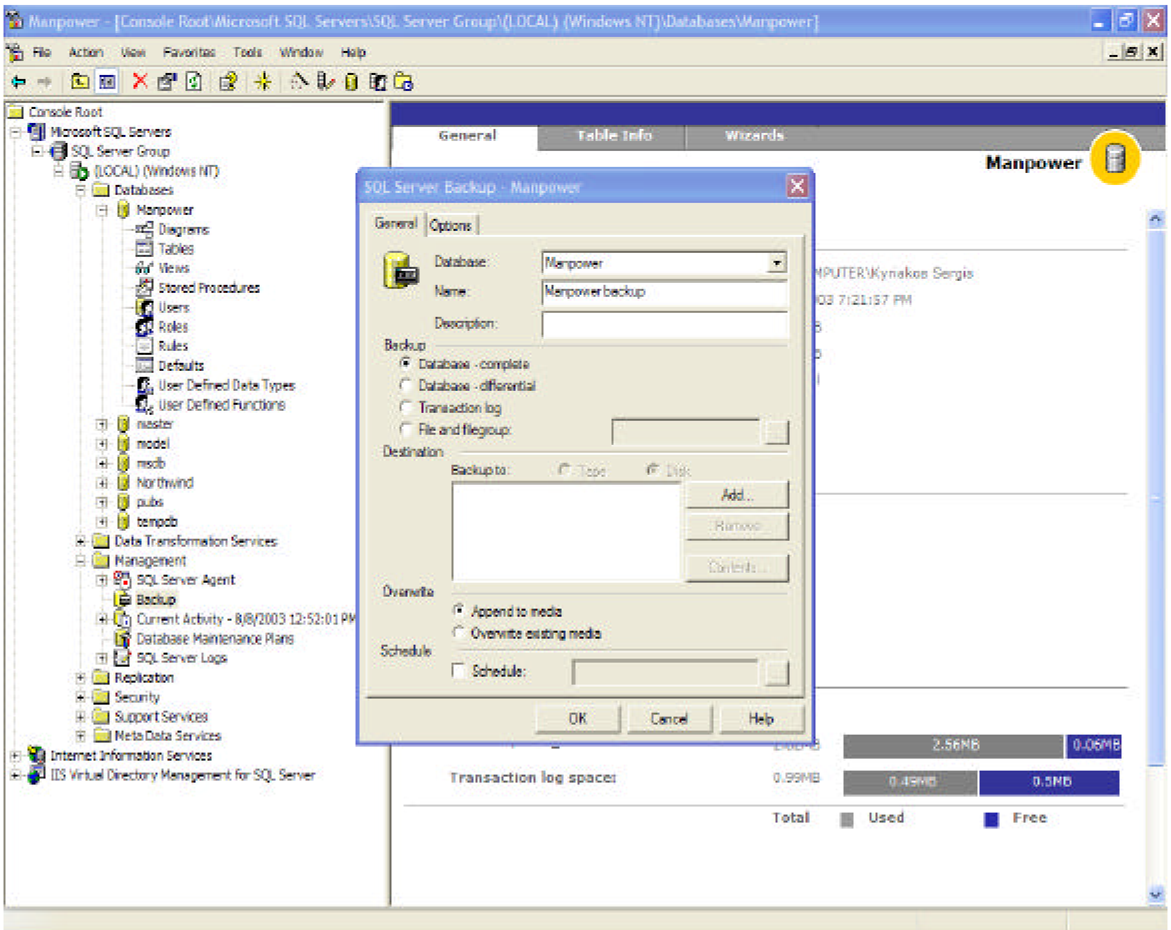


Figure 113. Backup-Manpower Database.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSIONS

The purpose of this thesis was to identify and analyze the requirements and develop a prototype web site for Manpower Database and Website. This research combined with the author's experience as a Greek Naval Officer provided the foundation for the detailed presentation of functional requirements and system architecture for the Manpower Database and Website. Once the requirements and architecture were defined, an operational database and web site prototype were developed. Having fulfilled the goal of the thesis, the purpose of this chapter is to present some conclusions, recommendations, and suggestions for further work regarding our analysis and the development and deployment of the Manpower Database and Website.

Currently, the Department of Personnel is following a rather old fashioned procedure to select an officer for a specific job. It is using proprietary systems like desktop computers, which have W2K Professional as their operating systems. Based on the needs of the Navy the DoP examines the jobs and their requirements, including the qualifications and credentials of the officers. It then assigns a job to an officer trying to find the best match between them. In this thesis a detailed system and user functional requirements are defined, along with a multi-dimensional decision algorithm for matching jobs with officers.

The final Manpower Website must be able to handle multi-step transactions. The system architecture presented in this thesis should be scalable to an enterprise-wide solution. Also, in order to develop a working prototype, specific software technologies had to be selected. The assumption of a Windows NT/2000/XP network environment, the selection of the IIS-5 Web Server and SQL Server 2000 database and the selection of the Macromedia Dreamweaver MX as design toolS forced certain design decisions in the construction of the prototype. Lastly, the programming used to develop the prototype was based on the efforts of a single, relatively inexperienced individual. Due to the magnitude and impact of this program, a team of experienced web programmers should develop the Manpower Database and Website. This statement, however, should not cause the reader to discount the potential worth of the prototype, since it provides a substantial start in this direction.

B. RECOMMENDATIONS

In the course of the research for this thesis, some important aspects of the Manpower Database and Website development have been discovered. These “lessons learned” should be carefully considered as of the Manpower Database and Website moves from concept to reality.

1. Technology Selection

A decision must be made regarding the specific software products to be used in the Manpower Database and Website. Our prototype used Microsoft products, and Macromedia Dreamweaver MX, which provide the benefits of integrated user accounts and system interoperability. Other systems may be more appropriate, however. For example Oracle products can be used or even open source software like MySQL and Linux. Whatever software products are selected, it is important to ensure that they are interoperable.

2. Definition of User Requirements

The User Requirements should be carefully defined in order to create the correct database schema and website functionality. Any late changes on the requirements can cause big problems, because it will be hard to undo all the work and redo it accordingly to the new requirements.

C. FURTHER WORK

This thesis has been developed in a single computer where a web server and database server have been installed. But this should not be the case for the implementation of the Manpower Database and Website. The following items describe some ideas for further work.

1. Component Distribution

It is preferable that the web server and the database server are not located in the same place for maintenance and security reasons. Investigation should be conducted to resolve these issues.

2. Security Analysis

This thesis addressed security issues in a rather general way, and incorporated standard web security methods such as Secure Socket Layer and access control through Windows permissions. However, due to the scope of the entire Manpower Database and Website development program, a thorough security analysis is recommended. Security personnel could conduct such an analysis, simulate attacks on the Manpower Database and Website prototype and recommend and/or construct programmatic security measures to incorporate into the Manpower Database and Website design.

3. Systems Architecture

A thorough analysis of the most appropriate system architecture for the entire Manpower Database and Website system is needed. A cost benefit analysis should be conducted to include server load, response time, code maintenance and upgrade, equipment and software costs, facility and manning requirements, web site and database administration procedures, database synchronization, and customer service.

4. Coefficient Weights and HValue Definition

The multi-criteria decision model uses several criteria such as credentials, language proficiency and officers' preference to determine the HValue as a number that expresses the suitability of an officer for a job. Also, the weights of each criterion determine the importance of each criterion and cause different HValues as they change. A thorough analysis of the computation and definition of the weights of each criterion should be performed according to the needs of the Greek Navy.

In summary, the prototype was developed virtually cost-free and can serve as a template for the development of a fully operational Manpower Database and Website; it can easily be scaled to the total solution. It is hoped that this thesis work will provide detailed insight for efforts in that direction so that the Manpower Database and Website may progress beyond conceptual planning to become a reality in the Greek Navy.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. TABLES

Table: ADDRESS			
Name	**Data Type**	**Size**	**Key**
CityOrTown	Char	50	Yes
Street	Char	50	Yes
Number	Char	10	Yes
Apartment	Char	10	Yes
ZIP	Char	10	Yes
ApplicantId	Char	10	Yes

Table: APPLICANT			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
FirstName	Char	30	
LastName	Char	30	
MiddleName	Char	30	
SeaTimeForRank	Float	8	
RankCode	Char	10	
SpecialtyCode	Char	10	
UserName	Char	50	
Password	Char	50	
EmailAddress	Char	50	
DetailerCheck	Bit	1	
DetailerPassword	Char	50	

Table: APPLICANT CREDENTIALS			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
CredentialsId	Char	10	Yes
CredentialsGrade	Int	4	

Table: APPLICANT LANGUAGE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
LanguageCode	Char	10	Yes
LanguageDegree	Float	8	

Table: APPLICANT PREFERENCE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
PreferenceApplicant	Int	4	

Table: ASSIGNED APPLICANTS			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes

Table: ASSIGNMENT			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	
PlaceCode	Char	10	
ReportDate	Datetime	8	
DetachDate	Datetime	8	

Table: COEFFICIENT			
Name	**Data Type**	**Size**	**Key**
CoefficientId	Char	30	Yes
CoefficientValue	Int	4	

Table: COMMAND			
Name	**Data Type**	**Size**	**Key**
CommandCode	Char	10	Yes
CommandName	Char	50	
UserName	Char	50	
Password	Char	50	

Table: COMMAND PREFERENCE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
CommandCode	Char	10	
PreferenceCommand	Int	4	

Table: COUNTER			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
Counter	Int	4	

Table: CREDENTIALS			
Name	**Data Type**	**Size**	**Key**
CredentialsId	Char	10	Yes
CredentialsName	Char	30	

Table: DELETED JOBS			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes

Table: DELETED JOBS MANIPULATE			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes

Table: ESTIMATE FUNCTION RESULT			
Name	**Data Type**	**Size**	**Key**
Result	Float	8	Yes

Table: EXPERIENCE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
Experience	Float	8	

Table: H			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
HValue	Float	8	

Table: JOB			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
JobName	Char	30	
ExperienceRequired	Float	8	
Priority	Int	4	

Table: JOB CREDENTIALS			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
CredentialsId	Char	10	Yes
CredentialsGrade	Int	4	

Table: JOB LANGUAGE			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
LanguageCode	Char	10	Yes
LanguageDegree	Float	8	

Table: JOB PLACE			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes

Table: JOB QUALIFICATION			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
QualificationCode	Char	10	Yes

Table: JOB RANK			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
RankCode	Char	10	Yes

Table: JOB SPECIALTY			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
SpecialtyCode	Char	10	Yes

Table: LANGUAGE			
Name	**Data Type**	**Size**	**Key**
LanguageCode	Char	10	Yes
LanguageName	Char	50	

Table: MANIPULATE SOLUTION			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
MAXValue	Float	8	

Table: MAX VALUE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
MAXValue	Float	8	

Table: MAX VALUE ALL JOBS			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
MAXValue	Float	8	

Table: MEAN VALUE			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
MeanValue	Float	8	

Table: MEAN VALUE APPLICANTS			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
MINValue	Float	8	

Table: MULTIPLE MAX VALUES			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
Counter	Int	4	

Table: ONE MAX VALUE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
Counter	Int	4	

Table: PHONE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
HomePhoneNumber	Char	30	Yes
CellPhoneNumber	Char	30	Yes
OtherPhoneNumber	Char	30	Yes

Table: PLACE			
Name	**Data Type**	**Size**	**Key**
PlaceCode	Char	10	Yes
PlaceName	Char	50	
PlaceImage	Char	10	
CommandCode	Char	10	

Table: PRIORITY			
Name	**Data Type**	**Size**	**Key**
JobId	Char	10	Yes
PlaceCode	Char	10	Yes
Priority	Int	4	
Counter	Int	4	
Flag	Bit	1	

Table: QUALIFICATION			
Name	**Data Type**	**Size**	**Key**
QualificationCode	Char	10	Yes
QualificationName	Char	50	

Table: QUALIFICATION APPLICANT			
Name	**Data Type**	**Size**	**Key**
QualificationCode	Char	10	Yes
ApplicantId	Char	10	Yes

Table: RANK			
Name	**Data Type**	**Size**	**Key**
RankCode	Char	10	Yes
RankName	Char	30	
TimeSeaService	Float	8	

Table: SAME MAX VALUE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes

Table: SPECIALTY			
Name	**Data Type**	**Size**	**Key**
SpecialtyCode	Char	10	Yes
SpecialtyName	Char	50	

Table: UNASSIGNED APPLICANTS			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes

Table: UNASSIGNED APPLICANTS MANIPULATE			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes

Table: USED APPLICANTS			
Name	**Data Type**	**Size**	**Key**
ApplicantId	Char	10	Yes
JobId	Char	10	Yes
PlaceCode	Char	10	Yes

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. STORED PROCEDURES

Name: AcceptSolutionFromManipulateSolutionTable

```
CREATE PROCEDURE ksergis.AcceptSolutionFromManipulateSolutionTable AS

DELETE FROM ASSIGNMENT

INSERT INTO ASSIGNMENT
SELECT JobId, PlaceCode, ApplicantId, NULL, NULL
FROM MANIPULATE_SOLUTION
GO
```

Name: AcceptSolutionFromMAXTable

```
CREATE PROCEDURE ksergis.AcceptSolutionFromMAXTable AS

DELETE FROM ASSIGNMENT

INSERT INTO ASSIGNMENT
SELECT JobId, PlaceCode, ApplicantId, NULL, NULL
FROM MAX_VALUE
GO
```

Name: CheckApplicantIdCredentialsId

```
CREATE PROCEDURE ksergis.CheckApplicantIdCredentialsId (@ApplicantId
char(10), @CredentialsId char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_CREDENTIALS WHERE
ApplicantId = @ApplicantId AND CredentialsId = @CredentialsId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantIdLanguageCode

```
CREATE PROCEDURE ksergis.CheckApplicantIdLanguageCode (@ApplicantId
```

```
char(10), @LanguageCode char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_LANGUAGE WHERE ApplicantId =
@ApplicantId AND LanguageCode = @LanguageCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantIdOnApplicantCredentials

```
CREATE PROCEDURE ksergis.CheckApplicantIdOnApplicantCredentials
(@ApplicantId char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_CREDENTIALS WHERE
ApplicantId = @ApplicantId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantIdOnApplicantLanguage

```
CREATE PROCEDURE ksergis.CheckApplicantIdOnApplicantLanguage
(@ApplicantId char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_LANGUAGE WHERE ApplicantId =
@ApplicantId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
```

```
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantIdOnQualificationsApplicant

```
CREATE PROCEDURE ksergis.CheckApplicantIdOnQualificationsApplicant
(@ApplicantId char(10))
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION_APPLICANT WHERE
ApplicantId = @ApplicantId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantIdQualificationCode

```
CREATE PROCEDURE ksergis.CheckApplicantIdQualificationCode (@ApplicantId
char(10), @QualificationCode char(10))
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION_APPLICANT WHERE
ApplicantId = @ApplicantId AND QualificationCode = @QualificationCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantPreferenceExists

```
CREATE PROCEDURE ksergis.CheckApplicantPreferenceExists (@ApplicantId
char(10))
AS
```

```
IF EXISTS(SELECT 'True' FROM APPLICANT_PREFERENCE WHERE ApplicantId
= @ApplicantId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantsExist

```
CREATE PROCEDURE ksergis.CheckApplicantsExist
AS
IF EXISTS(SELECT 'True' FROM APPLICANT)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckApplicantSuitable

```
CREATE PROCEDURE ksergis.CheckApplicantSuitable (@ApplicantId char(10)) AS

DECLARE @Rank int
DECLARE @Specialty int
DECLARE @Qualifications int
DECLARE @JobId char(10)
DECLARE @JobName char(30)

CREATE TABLE #SUITABLE_JOBS
(
  JobId char(10) PRIMARY KEY,
  JobName char(30)
)

DECLARE JobCursor CURSOR FOR
```

```
SELECT JobId, JobName
FROM JOB

OPEN JobCursor
FETCH NEXT FROM JobCursor
INTO @JobId, @JobName
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @Rank = ksergis.dec_Rank @JobId, @ApplicantId
		EXEC @Specialty = ksergis.dec_Specialty @JobId, @ApplicantId
		EXEC @Qualifications = ksergis.dec_Qualifications @JobId,
@ApplicantId

		IF @Rank = 1 AND @Specialty =1 AND @Qualifications = 1
		BEGIN
			INSERT INTO #SUITABLE_JOBS
			VALUES (@JobId, @JobName)
		END
	END
	FETCH NEXT FROM JobCursor
	INTO @JobId, @JobName
END

CLOSE JobCursor
DEALLOCATE JobCursor

SELECT *
FROM #SUITABLE_JOBS
GO
```

Name: CheckCoeffitientExists

```
CREATE PROCEDURE ksergis.CheckCoeffitientExists (@CoefficientId char(30))
AS
IF EXISTS(SELECT 'True' FROM COEFFICIENT WHERE CoefficientId =
@CoefficientId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
```

```
END
GO
```

Name: CheckCommandPreferenceExists

```
CREATE PROCEDURE ksergis.CheckCommandPreferenceExists (@CommandCode char(10))
AS
IF EXISTS(SELECT 'True' FROM COMMAND_PREFERENCE WHERE CommandCode = @CommandCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckCommandsExist

```
CREATE PROCEDURE ksergis.CheckCommandsExist
AS
IF EXISTS(SELECT 'True' FROM COMMAND)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckCredentialsExist

```
CREATE PROCEDURE ksergis.CheckCredentialsExist
AS
IF EXISTS(SELECT 'True' FROM CREDENTIALS)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
```

```
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckCredentialsId

CREATE PROCEDURE ksergis.CheckCredentialsId (@CredentialsId char(10))
AS
IF EXISTS(SELECT 'True' FROM CREDENTIALS WHERE CredentialsId = @CredentialsId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckCredentialsName

CREATE PROCEDURE ksergis.CheckCredentialsName (@CredentialsName char(50))
AS
IF EXISTS(SELECT 'True' FROM CREDENTIALS WHERE CredentialsName = @CredentialsName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckDateExist
CREATE PROCEDURE ksergis.CheckDateExist (@ApplicantId char(10)) AS IF EXISTS(SELECT 'True' FROM ASSIGNMENT WHERE ApplicantId =

```
@ApplicantId AND ((ReportDate IS NOT NULL) OR (DetachDate IS NOT NULL)))
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckDetailerPassword

```
CREATE PROCEDURE ksergis.CheckDetailerPassword (@ApplicantId char(10),
@DetailerPassword char(50))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT WHERE ApplicantId = @ApplicantId
AND DetailerPassword = @DetailerPassword)
BEGIN
  --This means it is correct, return it to ASP and tell us
  SELECT 'The Detailer is authenticated'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'The Detailer is not authenticated'
END
GO
```

Name: CheckExperienceExist

```
CREATE PROCEDURE ksergis.CheckExperienceExist (@JobId char(10),
@ApplicantId char(10))
AS
IF EXISTS(SELECT 'True' FROM EXPERIENCE WHERE JobId = @JobId AND
ApplicantId = @ApplicantId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
```

```
GO
```

Name: CheckJobId

```
CREATE PROCEDURE ksergis.CheckJobId (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdCredentialsId

```
CREATE PROCEDURE ksergis.CheckJobIdCredentialsId (@JobId char(10),
@CredentialsId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_CREDENTIALS WHERE JobId = @JobId
AND CredentialsId = @CredentialsId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdJobName

```
CREATE PROCEDURE ksergis.CheckJobIdJobName (@JobId char(10), @JobName
char(30))
AS
IF EXISTS(SELECT 'True' FROM JOB WHERE JobId = @JobId OR JobName =
@JobName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
```

```
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdLanguageCode

```
CREATE PROCEDURE ksergis.CheckJobIdLanguageCode (@JobId char(10), @LanguageCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_LANGUAGE WHERE JobId = @JobId AND LanguageCode = @LanguageCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnApplicantPreference

```
CREATE PROCEDURE ksergis.CheckJobIdOnApplicantPreference (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_PREFERENCE WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnCommandPreference

```
CREATE PROCEDURE ksergis.CheckJobIdOnCommandPreference (@JobId char(10))
```

```
AS
IF EXISTS(SELECT 'True' FROM COMMAND_PREFERENCE WHERE JobId =
@JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnJobCredentials

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobCredentials (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_CREDENTIALS WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnJobLanguage

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobLanguage (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_LANGUAGE WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnJobPlace

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobPlace (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_PLACE WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnJobQualification

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobQualification (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_QUALIFICATION WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdOnJobRank

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobRank (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_RANK WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
```

GO

Name: CheckJobIdOnJobSpecialty

```
CREATE PROCEDURE ksergis.CheckJobIdOnJobSpecialty (@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_SPECIALTY WHERE JobId = @JobId)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdPlaceCode

```
CREATE PROCEDURE ksergis.CheckJobIdPlaceCode (@JobId char(10), @PlaceCode
char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_PLACE WHERE JobId = @JobId AND
PlaceCode = @PlaceCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdPlaceCodeOnApplicantPreference

```
CREATE PROCEDURE ksergis.CheckJobIdPlaceCodeOnApplicantPreference (@JobId
char(10), @PlaceCode char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT_PREFERENCE WHERE JobId =
@JobId AND PlaceCode = @PlaceCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
```

```
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdPlaceCodeOnCommandPreference

```
CREATE PROCEDURE ksergis.CheckJobIdPlaceCodeOnCommandPreference (@JobId
char(10), @PlaceCode char(10))
AS
IF EXISTS(SELECT 'True' FROM COMMAND_PREFERENCE WHERE JobId =
@JobId AND PlaceCode = @PlaceCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdQualificationCode

```
CREATE PROCEDURE ksergis.CheckJobIdQualificationCode (@JobId char(10),
@QualificationCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_QUALIFICATION WHERE JobId = @JobId
AND QualificationCode = @QualificationCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdRankCode

```
CREATE PROCEDURE ksergis.CheckJobIdRankCode (@JobId char(10), @RankCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_RANK WHERE JobId = @JobId AND RankCode = @RankCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobIdSpecialtyCode

```
CREATE PROCEDURE ksergis.CheckJobIdSpecialtyCode (@JobId char(10), @SpecialtyCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_SPECIALTY WHERE JobId = @JobId AND SpecialtyCode = @SpecialtyCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobName

```
CREATE PROCEDURE ksergis.CheckJobName (@JobName char(30))
AS
IF EXISTS(SELECT 'True' FROM JOB WHERE JobName = @JobName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
```

```
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckJobsExist

CREATE PROCEDURE ksergis.CheckJobsExist
AS
IF EXISTS(SELECT 'True' FROM JOB)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckLanguageCode

CREATE PROCEDURE ksergis.CheckLanguageCode (@LanguageCode char(10))
AS
IF EXISTS(SELECT 'True' FROM LANGUAGE WHERE LanguageCode = @LanguageCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckLanguageName

CREATE PROCEDURE ksergis.CheckLanguageName (@LanguageName char(50))
AS
IF EXISTS(SELECT 'True' FROM LANGUAGE WHERE LanguageName = @LanguageName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'

```
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckLanguagesExist

CREATE PROCEDURE ksergis.CheckLanguagesExist
AS
IF EXISTS(SELECT 'True' FROM LANGUAGE)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckPlacesExist

CREATE PROCEDURE ksergis.CheckPlacesExist
AS
IF EXISTS(SELECT 'True' FROM PLACE)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO

Name: CheckPreference
CREATE PROCEDURE ksergis.CheckPreference (@ApplicantId varchar(10), @PreferenceApplicant varchar(4), @PlaceCode varchar(10), @JobId varchar(10)) AS IF EXISTS(SELECT 'True' FROM APPLICANT_PREFERENCE WHERE ApplicantId

```
= @ApplicantId AND (PreferenceApplicant = @PreferenceApplicant OR (JobId =
@JobId AND PlaceCode = @PlaceCode)))
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This preference already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This preference does not exist!'
END
GO
```

Name: CheckPreferenceCommand

```
CREATE PROCEDURE ksergis.CheckPreferenceCommand (@CommandCode
char(10), @ApplicantId char(10), @PreferenceCommand char(4), @PlaceCode char(10),
@JobId char(10))
AS
IF EXISTS(SELECT 'True' FROM COMMAND_PREFERENCE WHERE
CommandCode = @CommandCode AND JobId = @JobId AND PlaceCode =
@PlaceCode AND ( PreferenceCommand = @PreferenceCommand OR ApplicantId =
@ApplicantId))
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This preference already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This preference does not exist!'
END
GO
```

Name: CheckQualificationCode

```
CREATE PROCEDURE ksergis.CheckQualificationCode (@QualificationCode
char(10))
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION WHERE QualificationCode =
@QualificationCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
```

```
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckQualificationName

```
CREATE PROCEDURE ksergis.CheckQualificationName (@QualificationName char(50))
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION WHERE QualificationName = @QualificationName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckQualificationsExist

```
CREATE PROCEDURE ksergis.CheckQualificationsExist
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckRankCode

```
CREATE PROCEDURE ksergis.CheckRankCode (@RankCode char(10))
AS
IF EXISTS(SELECT 'True' FROM RANK WHERE RankCode = @RankCode)
BEGIN
```

```
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckRankName

```
CREATE PROCEDURE ksergis.CheckRankName (@RankName char(30))
AS
IF EXISTS(SELECT 'True' FROM RANK WHERE RankName = @RankName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckRanksExist

```
CREATE PROCEDURE ksergis.CheckRanksExist
AS
IF EXISTS(SELECT 'True' FROM RANK)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckSpecialtiesExist

```
CREATE PROCEDURE ksergis.CheckSpecialtiesExist
AS
```

```
IF EXISTS(SELECT 'True' FROM SPECIALTY)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckSpecialtyCode

```
CREATE PROCEDURE ksergis.CheckSpecialtyCode (@SpecialtyCode char(10))
AS
IF EXISTS(SELECT 'True' FROM SPECIALTY WHERE SpecialtyCode = @SpecialtyCode)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckSpecialtyName

```
CREATE PROCEDURE ksergis.CheckSpecialtyName (@SpecialtyName char(50))
AS
IF EXISTS(SELECT 'True' FROM SPECIALTY WHERE SpecialtyName = @SpecialtyName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckSuitableApplicantsOnJob

```
CREATE PROCEDURE ksergis.CheckSuitableApplicantsOnJob (@JobId char(10)) AS

DECLARE @Rank int
DECLARE @Specialty int
DECLARE @Qualifications int
DECLARE @ApplicantId char(10)
DECLARE @FirstName char(30)
DECLARE @LastName char(30)

CREATE TABLE #SUITABLE_APPLICANTS
(
  ApplicantId char(10) PRIMARY KEY,
  FirstName char(30),
  LastName char(30)
)

DECLARE ApplicantCursor CURSOR FOR
SELECT  ApplicantId, FirstName, LastName
FROM APPLICANT

OPEN ApplicantCursor
FETCH NEXT FROM ApplicantCursor
INTO @ApplicantId, @FirstName, @LastName
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @Rank = ksergis.dec_Rank @JobId, @ApplicantId
		EXEC @Specialty = ksergis.dec_Specialty @JobId, @ApplicantId
		EXEC @Qualifications = ksergis.dec_Qualifications @JobId,
@ApplicantId

		IF @Rank = 1 AND @Specialty =1 AND @Qualifications = 1
		BEGIN
			INSERT INTO #SUITABLE_APPLICANTS
			VALUES (@ApplicantId, @FirstName, @LastName)
		END
	END
	FETCH NEXT FROM ApplicantCursor
	INTO @ApplicantId, @FirstName, @LastName
END

CLOSE ApplicantCursor
DEALLOCATE ApplicantCursor
```

```
SELECT *
FROM #SUITABLE_APPLICANTS
GO
```

Name: CheckUserName

```
CREATE PROCEDURE ksergis.CheckUserName (@UserName varchar(50))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT WHERE UserName = @UserName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: CheckUserNameCommand

```
CREATE PROCEDURE ksergis.CheckUserNameCommand (@UserName varchar(50))
AS
IF EXISTS(SELECT 'True' FROM COMMAND WHERE UserName = @UserName)
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'This record already exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'This record does not exist!'
END
GO
```

Name: dec_CheckHValueExists

```
CREATE PROCEDURE ksergis.dec_CheckHValueExists (@Counter int)
AS

DECLARE @JobId char(10)
DECLARE @JobId1 char(10)
DECLARE @PlaceCode char(10)
DECLARE @PlaceCode1 char(10)
```

```
DECLARE PriorityCursor CURSOR FOR
SELECT JobId, PlaceCode, Counter
FROM PRIORITY
WHERE Counter = @Counter

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @JobId1 = @JobId
		SET @PlaceCode1 = @PlaceCode
	END
	FETCH NEXT FROM PriorityCursor
	INTO @JobId, @PlaceCode, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor

IF EXISTS(SELECT HValue FROM H WHERE JobId = @JobId1 AND PlaceCode =
@PlaceCode1 AND HValue IS NOT NULL AND
					ApplicantId NOT IN (SELECT ApplicantId
FROM USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode =
@PlaceCode1) AND
					ApplicantId NOT IN (SELECT ApplicantId
FROM ASSIGNED_APPLICANTS))
	RETURN 1
ELSE
	RETURN 0
GO
```

Name: dec_ CheckHValueNotNull

```
CREATE PROCEDURE ksergis.dec_CheckHValueNotNull (@JobId char(10),
@PlaceCode char(10), @ApplicantId  char(10)) AS

DECLARE @HValue float

SET @HValue = (SELECT HValue FROM H WHERE JobId = @JobId AND PlaceCode
= @PlaceCode AND ApplicantId = @ApplicantId)

IF @HValue IS NOT NULL
```

```
BEGIN
  --This means it exists, return it to ASP and tell us
  SELECT 'HValue exists!'
END
ELSE
BEGIN
  --This means it does not exist, return it to ASP and tell us
  SELECT 'HValue does not exist!'
END
GO
```

Name: dec_ ComputeMaxValue

```
CREATE PROCEDURE ksergis.dec_ComputeMaxValue (@Counter int)
AS

DECLARE @JobId char(10)
DECLARE @JobId1 char(10)
DECLARE @PlaceCode char(10)
DECLARE @PlaceCode1 char(10)
DECLARE @CountEqualMaxValues int

DECLARE PriorityCursor CURSOR FOR
SELECT JobId, PlaceCode, Counter
FROM PRIORITY
WHERE Counter = @Counter

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @JobId1 = @JobId
		SET @PlaceCode1 = @PlaceCode
	END
	FETCH NEXT FROM PriorityCursor
	INTO @JobId, @PlaceCode, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor

DECLARE @MAXValue float
DECLARE @ApplicantId char(10)
```

```
DECLARE @ApplicantId1 char(10)

SET @MAXValue = (SELECT MAX(HValue) FROM H WHERE JobId = @JobId1
AND PlaceCode = @PlaceCode1 AND HValue IS NOT NULL AND
                                                            ApplicantId NOT IN (SELECT
ApplicantId FROM USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode
= @PlaceCode1) AND
                                                            ApplicantId NOT IN (SELECT
ApplicantId FROM ASSIGNED_APPLICANTS))

SET @CountEqualMaxValues = (      SELECT count(ApplicantId)
                              FROM H
                              WHERE JobId = @JobId1 AND PlaceCode =
@PlaceCode1 AND HValue = @MAXValue AND
                              ApplicantId NOT IN (SELECT ApplicantId FROM
USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
AND
                              ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS))

IF @CountEqualMaxValues > 1
        EXEC @ApplicantId1 = ksergis.dec_FindMaxValue @JobId1, @PlaceCode1,
@MAXValue
ELSE
BEGIN
        DECLARE HCursor CURSOR FOR
        SELECT ApplicantId
        FROM H
        WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1 AND HValue =
@MAXValue AND
                ApplicantId NOT IN (SELECT ApplicantId FROM
USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
AND
                ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS)

        OPEN HCursor
        FETCH NEXT FROM HCursor
        INTO @ApplicantId
        WHILE @@FETCH_STATUS <> -1
        BEGIN
                IF @@FETCH_STATUS <> -2
                SET @ApplicantId1 = @ApplicantId
                BREAK
                FETCH NEXT FROM HCursor
                INTO @ApplicantId
```

```
	END

	CLOSE HCursor
	DEALLOCATE HCursor
END

PRINT 'MAXValue'
PRINT @MAXValue
PRINT 'ApplicantId1'
PRINT @ApplicantId1

UPDATE MAX_VALUE
SET ApplicantId = @ApplicantId1, MAXValue = @MAXValue
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1

INSERT INTO ASSIGNED_APPLICANTS
SELECT ApplicantId
FROM APPLICANT
WHERE ApplicantId = @ApplicantId1
GO
```

Name: dec_ ComputeMeanValue

```
CREATE PROCEDURE ksergis.dec_ComputeMeanValue
AS

DELETE FROM MEAN_VALUE

INSERT INTO MEAN_VALUE
SELECT JobId, PlaceCode, NULL
FROM JOB_PLACE

DECLARE @JobId char(10)
DECLARE @PlaceCode char(10)
DECLARE @MeanValue float

DECLARE MeanValueCursor CURSOR FOR
SELECT JobId, PlaceCode, MeanValue
FROM MEAN_VALUE

OPEN MeanValueCursor
FETCH NEXT FROM MeanValueCursor
INTO @JobId, @PlaceCode, @MeanValue
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
```

```
        BEGIN

                DECLARE @ApplicantId char(10)
                DECLARE @HValue float
                DECLARE @SUM float
                DECLARE @COUNT int

                DECLARE HCursor CURSOR FOR
                SELECT JobId, ApplicantId, PlaceCode, HValue
                FROM H
                WHERE JobId = @JobId AND PlaceCode = @PlaceCode

                SET @SUM = 0
                SET @COUNT = 0

                OPEN HCursor
                FETCH NEXT FROM HCursor
                INTO @JobId, @ApplicantId, @PlaceCode, @HValue
                WHILE @@FETCH_STATUS <> -1
                BEGIN
                        IF @@FETCH_STATUS <> -2
                        BEGIN
                                IF @HValue IS NOT NULL
                                BEGIN
                                        SET @SUM = @SUM + @HValue
                                        SET @COUNT = @COUNT + 1
                                END
                        END
                        FETCH NEXT FROM HCursor
                        INTO @JobId, @ApplicantId, @PlaceCode, @HValue
                END

                CLOSE HCursor
                DEALLOCATE HCursor

                IF @SUM <> 0
                        UPDATE MEAN_VALUE
                        SET MeanValue = @SUM / @COUNT
                        WHERE JobId = @JobId AND PlaceCode= @PlaceCode

        END
        FETCH NEXT FROM MeanValueCursor
        INTO @JobId, @PlaceCode, @MeanValue
END

CLOSE MeanValueCursor
```

```
DEALLOCATE MeanValueCursor
GO
```

Name: dec_COUNTER_Fill

```
CREATE PROCEDURE ksergis.dec_COUNTER_Fill
AS

DELETE FROM COUNTER

INSERT INTO COUNTER
SELECT JobId, PlaceCode, Counter
FROM PRIORITY
GO
```

Name: dec_ CountPriorityRecords

```
CREATE PROCEDURE ksergis.dec_CountPriorityRecords
AS
DECLARE @Count int

SET @Count = (SELECT Count (*) FROM PRIORITY)

RETURN @Count
GO
```

Name: dec_ Credentials

```
CREATE PROCEDURE ksergis.dec_Credentials (@JobId char(10), @ApplicantId
char(10))
AS
DECLARE @CredentialsGrade1 float
DECLARE @CredentialsGrade2 float
DECLARE @CredentialsId char(10)
DECLARE @SUM1 float
DECLARE @SUM2 float
DECLARE @ANS float
DECLARE @Count int

SET @SUM1 = 0
SET @SUM2 = 0
SET @Count = 0

DECLARE CredentialsCursor CURSOR FOR
SELECT JobId, CredentialsId
FROM JOB_CREDENTIALS
```

```
WHERE JobId = @JobId

OPEN CredentialsCursor
FETCH NEXT FROM CredentialsCursor
INTO @JobId, @CredentialsId
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @CredentialsGrade1 = ksergis.dec_Credentials1 @ApplicantId,
@CredentialsId
		EXEC @CredentialsGrade2 = ksergis.dec_Credentials2 @JobId,
@CredentialsId
		SET @SUM1 = @SUM1 +@CredentialsGrade1
		SET @SUM2 = @SUM2 +@CredentialsGrade2
		SET @Count = @Count +1
	END
	FETCH NEXT FROM CredentialsCursor
	INTO @JobId, @CredentialsId
END

CLOSE CredentialsCursor
DEALLOCATE CredentialsCursor

IF @SUM1 < @SUM2
	SET @ANS = 0
ELSE
BEGIN
	IF @Count * 10 = @SUM2
		SET @ANS = 1
	ELSE
		SET @ANS = ((@SUM1 - @SUM2) * 9 / ((@Count * 10) - @SUM2)) +
1
END

RETURN @ANS
GO
```

Name: dec_ Credentials1

```
CREATE PROCEDURE ksergis.dec_Credentials1 (@ApplicantId char(10),
@CredentialsId char(10))
AS
DECLARE @CredentialsGrade int

IF EXISTS (SELECT CredentialsGrade FROM APPLICANT_CREDENTIALS WHERE
```

```
ApplicantId = @ApplicantId AND CredentialsId = @CredentialsId)
        SET @CredentialsGrade = (SELECT CredentialsGrade FROM
APPLICANT_CREDENTIALS WHERE ApplicantId = @ApplicantId AND
CredentialsId = @CredentialsId)
ELSE
        SET @CredentialsGrade = 0
RETURN @CredentialsGrade
GO
```

Name: dec_ Credentials2
CREATE PROCEDURE ksergis.dec_Credentials2 (@JobId char(10), @CredentialsId char(10)) AS DECLARE @CredentialsGrade int SET @CredentialsGrade = (SELECT CredentialsGrade FROM JOB_CREDENTIALS WHERE JobId = @JobId AND CredentialsId = @CredentialsId) RETURN @CredentialsGrade GO

Name: dec_ Delete_Job_Manipulate
CREATE PROCEDURE ksergis.dec_Delete_Job_Manipulate (@JobId char(10), @PlaceCode char(10)) AS DECLARE @ApplicantId char(10) SET @ApplicantId = (SELECT ApplicantId FROM MANIPULATE_SOLUTION WHERE JobId = @JobId AND PlaceCode = @PlaceCode) DELETE FROM MANIPULATE_SOLUTION WHERE JobId = @JobId AND PlaceCode = @PlaceCode INSERT INTO UNASSIGNED_APPLICANTS_MANIPULATE VALUES (@ApplicantId) INSERT INTO DELETED_JOBS_MANIPULATE VALUES (@JobId, @PlaceCode) GO

Name: dec_ DELETED_JOBS_MANIPULATE_DeleteRecord
CREATE PROCEDURE ksergis.dec_DELETED_JOBS_MANIPULATE_DeleteRecord (@JobId char(10), @PlaceCode char(10)) AS DELETE FROM DELETED_JOBS_MANIPULATE WHERE JobId = @JobId AND PlaceCode = @PlaceCode

```
GO
```

Name: dec_ DELETED_JOBS_MANIPULATE_Fill

```
CREATE PROCEDURE ksergis.dec_DELETED_JOBS_MANIPULATE_Fill AS
DELETE FROM DELETED_JOBS_MANIPULATE

INSERT INTO DELETED_JOBS_MANIPULATE
SELECT *
FROM DELETED_JOBS
GO
```

Name: dec_ DeleteEmptyJobs

```
CREATE PROCEDURE ksergis.dec_DeleteEmptyJobs
AS

DECLARE @JobId char(10)
DECLARE @ApplicantId char(10)
DECLARE @PlaceCode char(10)
DECLARE @Counter int
DECLARE @HValue float

DECLARE PriorityCursor CURSOR FOR
SELECT JobId, PlaceCode, Counter
FROM PRIORITY

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		IF NOT EXISTS (SELECT 'True' FROM H WHERE JobId = @JobId
AND PlaceCode= @PlaceCode AND HValue IS NOT NULL)
		BEGIN
			INSERT INTO DELETED_JOBS
			SELECT JobId, PlaceCode
			FROM PRIORITY
			WHERE JobId = @JobId  AND PlaceCode= @PlaceCode

			DELETE FROM PRIORITY
			WHERE JobId = @JobId  AND PlaceCode= @PlaceCode

			UPDATE PRIORITY
```

```
                        SET Counter = Counter - 1
                        WHERE Counter > @Counter
                END
        END
        FETCH NEXT FROM PriorityCursor
        INTO @JobId, @PlaceCode, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor
GO
```

Name: dec_ DeleteJob
CREATE PROCEDURE ksergis.dec_DeleteJob AS DECLARE @Counter int SET @Counter = (SELECT MIN(Counter) FROM PRIORITY WHERE Flag = '0') INSERT INTO DELETED_JOBS SELECT JobId, PlaceCode FROM PRIORITY WHERE Counter = @Counter DELETE FROM PRIORITY WHERE Counter = @Counter UPDATE PRIORITY SET Counter = Counter - 1 WHERE Counter > @Counter GO

Name: dec_ DeleteJobUsedValues
CREATE PROCEDURE ksergis.dec_DeleteJobUsedValues (@Counter int) AS DECLARE @JobId char(10) DECLARE @JobId1 char(10) DECLARE @PlaceCode char(10) DECLARE @PlaceCode1 char(10) DECLARE PriorityCursor CURSOR FOR SELECT JobId, PlaceCode, Counter FROM PRIORITY

```
WHERE Counter = @Counter

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @JobId1 = @JobId
		SET @PlaceCode1 = @PlaceCode
	END
	FETCH NEXT FROM PriorityCursor
	INTO @JobId, @PlaceCode, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor

DELETE FROM USED_APPLICANTS
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1
GO
```

Name: dec_ EstimateFunction

```
CREATE PROCEDURE ksergis.dec_EstimateFunction AS

DECLARE @Priority1 int
DECLARE @Priority2 int
DECLARE @TotalValueMAXTable float
DECLARE @TotalValueManipulateTable float
DECLARE @Difference float
DECLARE @n_MAXTable int
DECLARE @n_ManipulateTable int
DECLARE @n_CounterTable int
DECLARE @SecondMaxValue float
DECLARE @MinValue float
DECLARE @MaxValue float
DECLARE @Factor float
DECLARE @Counter1 int
DECLARE @Counter2 int

DECLARE @JobId char(10)
DECLARE @PlaceCode char(10)
DECLARE @MAXValue1 float
DECLARE @MAXValue2 float
```

```
DECLARE @JobId1 char(10)
DECLARE @PlaceCode1 char(10)

SET @TotalValueMAXTable = 0
SET @TotalValueManipulateTable = 0

SET @n_MAXTable = (SELECT Count (*) FROM MAX_VALUE)
SET @n_ManipulateTable = (SELECT Count (*) FROM MANIPULATE_SOLUTION)
SET @n_CounterTable = (SELECT Count (*) FROM COUNTER)

SET @MaxValue = (SELECT max(HValue) FROM H WHERE HValue IS NOT NULL)
SET @MinValue = (SELECT min(HValue) FROM H WHERE HValue IS NOT NULL)

IF @MaxValue = @MinValue
	SET @Difference = 0
ELSE
BEGIN
	SET @SecondMaxValue = (SELECT max(HValue) FROM H WHERE HValue <
@MaxValue AND HValue IS NOT NULL)
	SET @Factor = 9/(@MaxValue - @SecondMaxValue)

	DECLARE MaxValueCursor CURSOR FOR
	SELECT JobId, PlaceCode, MAXValue
	FROM MAX_VALUE

	OPEN MaxValueCursor
	FETCH NEXT FROM MaxValueCursor
	INTO @JobId, @PlaceCode, @MAXValue1
	WHILE @@FETCH_STATUS <> -1
	BEGIN
		IF @@FETCH_STATUS <> -2
		BEGIN
			SET @Priority1 = (SELECT Counter FROM COUNTER WHERE
JobId = @JobId AND PlaceCode = @PlaceCode)

			SET @Counter1 = @Priority1 + 1
			WHILE @Counter1 <= @n_CounterTable
			BEGIN
				SET @JobId1 = (SELECT JobId FROM COUNTER
WHERE Counter = @Counter1)
				SET @PlaceCode1 = (SELECT PlaceCode FROM
COUNTER WHERE Counter = @Counter1)
				IF EXISTS (SELECT 'True' FROM MAX_VALUE
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
				BEGIN
```

```
                    SET @MAXValue2 = (SELECT MAXValue
FROM MAX_VALUE WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
                    SET @Priority2 = @Counter1
                    SET @TotalValueMAXTable =
@TotalValueMAXTable + log10((POWER(@Factor, (@n_CounterTable - @Priority1))
* @Priority1 * @MAXValue1) + (POWER(@Factor, (@n_CounterTable - @Priority2))
* @Priority2 * @MAXValue2))
                    BREAK
                END
                SET @Counter1 = @Counter1 + 1
            END
        END
        FETCH NEXT FROM MaxValueCursor
        INTO @JobId, @PlaceCode, @MAXValue1
    END

    CLOSE MaxValueCursor
    DEALLOCATE MaxValueCursor

    DECLARE ManipulateTableCursor CURSOR FOR
    SELECT JobId, PlaceCode, MAXValue
    FROM MANIPULATE_SOLUTION

    OPEN ManipulateTableCursor
    FETCH NEXT FROM ManipulateTableCursor
    INTO @JobId, @PlaceCode, @MAXValue1
    WHILE @@FETCH_STATUS <> -1
    BEGIN
        IF @@FETCH_STATUS <> -2
        BEGIN
            SET @Priority1 = (SELECT Counter FROM COUNTER WHERE
JobId = @JobId AND PlaceCode = @PlaceCode)

            SET @Counter1 = @Priority1 + 1
            WHILE @Counter1 <= @n_CounterTable
            BEGIN
                SET @JobId1 = (SELECT JobId FROM COUNTER
WHERE Counter = @Counter1)
                SET @PlaceCode1 = (SELECT PlaceCode FROM
COUNTER WHERE Counter = @Counter1)
                IF EXISTS (SELECT 'True' FROM
MANIPULATE_SOLUTION WHERE JobId = @JobId1 AND PlaceCode =
@PlaceCode1)
                BEGIN
                    SET @MAXValue2 = (SELECT MAXValue
FROM MANIPULATE_SOLUTION WHERE JobId = @JobId1 AND PlaceCode =
```

```
@PlaceCode1)
                                        SET @Priority2 = @Counter1
                                        SET @TotalValueManipulateTable =
@TotalValueManipulateTable + log10((POWER(@Factor, (@n_CounterTable -
@Priority1)) * @Priority1 * @MAXValue1) + (POWER(@Factor, (@n_CounterTable -
@Priority2)) * @Priority2 * @MAXValue2))
                                        BREAK
                                END
                                SET @Counter1 = @Counter1 + 1
                        END
                END
                FETCH NEXT FROM ManipulateTableCursor
                INTO @JobId, @PlaceCode, @MAXValue1
        END

        CLOSE ManipulateTableCursor
        DEALLOCATE ManipulateTableCursor

        SET @Difference = @TotalValueMAXTable - @TotalValueManipulateTable

print @TotalValueMAXTable
print @TotalValueManipulateTable
END

DELETE FROM ESTIMATE_FUNCTION_RESULT

INSERT INTO ESTIMATE_FUNCTION_RESULT
VALUES (@Difference)
GO
```

Name: dec_ Experience

```
CREATE PROCEDURE ksergis.dec_Experience (@JobId char(10), @ApplicantId
char(10))
AS
DECLARE @ExperienceRequired float
DECLARE @ExperienceYears float

SET @ExperienceYears = 0

SET @ExperienceRequired = (SELECT ExperienceRequired FROM JOB WHERE JobId
= @JobId)
IF (SELECT distinct(Experience) FROM EXPERIENCE WHERE ApplicantId =
@ApplicantId AND JobId = @JobId) IS NOT NULL
        SET @ExperienceYears = (SELECT distinct(Experience) FROM EXPERIENCE
WHERE ApplicantId = @ApplicantId AND JobId = @JobId)
```

```
IF @ExperienceYears < @ExperienceRequired
	RETURN 0
ELSE
	RETURN ((@ExperienceYears - @ExperienceRequired) * 9 / (15 - @ExperienceRequired)) + 1
GO
```

Name: dec_ FindMaxValue

```
CREATE PROCEDURE ksergis.dec_FindMaxValue (@JobId  char(10), @PlaceCode char(10), @MAXValue float)
AS
print 'inside findmaxvalue'
DECLARE @JobId1 char(10)
DECLARE @PlaceCode1 char(10)
DECLARE @JobId2 char(10)
DECLARE @PlaceCode2 char(10)
DECLARE @ApplicantId1 char(10)
DECLARE @ApplicantId2 char(10)
DECLARE @ApplicantId char(10)
DECLARE @Counter int
DECLARE @Counter1 int
DECLARE @Counter2 int
DECLARE @MinCount int
DECLARE @Temp int
DECLARE @Temp1 int
DECLARE @Temp2 int
DECLARE @C int
DECLARE @C1 int
DECLARE @C2 int
DECLARE @MultipleMaxValues int
DECLARE @Spot int
DECLARE @Length int
DECLARE @MAX float
DECLARE @MIN float
DECLARE @MAX1 float
DECLARE @MAX2 float
DECLARE @MIN1 float
DECLARE @HValue1 float
DECLARE @Eurika int
DECLARE @Flag int
DECLARE @MIN_VALUE_APPLICANTS_Length int
DECLARE @MULTIPLE_MAX_VALUES_Length int
DECLARE @ONE_MAX_VALUE_Length int
```

```
DELETE FROM SAME_MAX_VALUE
DELETE FROM MIN_VALUE_APPLICANTS
DELETE FROM MULTIPLE_MAX_VALUES
DELETE FROM ONE_MAX_VALUE

SET @Counter = (SELECT Counter FROM PRIORITY WHERE JobId = @JobId AND
PlaceCode = @PlaceCode)

DECLARE HCursor CURSOR FOR
SELECT ApplicantId
FROM H
WHERE JobId = @JobId AND PlaceCode = @PlaceCode AND HValue = @MAXValue
AND
        ApplicantId NOT IN (SELECT ApplicantId FROM USED_APPLICANTS
WHERE JobId = @JobId AND PlaceCode = @PlaceCode) AND
        ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS)

OPEN HCursor
FETCH NEXT FROM HCursor
INTO @ApplicantId
WHILE @@FETCH_STATUS <> -1
BEGIN
        IF @@FETCH_STATUS <> -2
        INSERT INTO SAME_MAX_VALUE
        VALUES (@ApplicantId)
        FETCH NEXT FROM HCursor
        INTO @ApplicantId
END

CLOSE HCursor
DEALLOCATE HCursor

EXEC @Length = ksergis.dec_CountPriorityRecords

SET @Eurika = 0
SET @Spot = 0

DECLARE SameMaxValueCursor CURSOR FOR
SELECT ApplicantId
FROM SAME_MAX_VALUE

OPEN SameMaxValueCursor
FETCH NEXT FROM SameMaxValueCursor
INTO @ApplicantId
WHILE @@FETCH_STATUS <> -1
```

```
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @Flag = 0
		SET @Counter1 = @Counter + 1
		WHILE @Counter1 <= @Length
		BEGIN
			SET @JobId1 = (SELECT JobId FROM PRIORITY WHERE
Counter = @Counter1)
			SET @PlaceCode1 = (SELECT PlaceCode FROM PRIORITY
WHERE Counter = @Counter1)
			SET @MAX1 = (SELECT max(HValue) FROM H WHERE JobId
= @JobId1 AND PlaceCode = @PlaceCode1 AND HValue IS NOT NULL AND
					ApplicantId NOT IN (SELECT ApplicantId FROM
USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
AND
					ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS))
			SET @HValue1 = (SELECT HValue FROM H WHERE JobId =
@JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId = @ApplicantId)
			IF @HValue1 = @MAX1
				SET @Flag = 1
			SET @Counter1 = @Counter1 + 1
		END

		IF @Flag = 0
		BEGIN
			SET @JobId2 = (SELECT JobId FROM PRIORITY WHERE
Counter = @Counter + 1)
			SET @PlaceCode2 = (SELECT PlaceCode FROM PRIORITY
WHERE Counter = @Counter + 1)
			SET @MIN1 = (SELECT HValue FROM H WHERE JobId =
@JobId2 AND PlaceCode = @PlaceCode2 AND ApplicantId = @ApplicantId)
			INSERT INTO MIN_VALUE_APPLICANTS
			VALUES (@ApplicantId, @MIN1)
		END
	END
	FETCH NEXT FROM SameMaxValueCursor
	INTO @ApplicantId
END

CLOSE SameMaxValueCursor
DEALLOCATE SameMaxValueCursor

SET @MIN_VALUE_APPLICANTS_Length = (SELECT count(*) FROM
MIN_VALUE_APPLICANTS)
```

```
IF @MIN_VALUE_APPLICANTS_Length > 0
BEGIN
        SET @Eurika = 1
        SET @MIN1 = (SELECT min(MINValue) FROM
MIN_VALUE_APPLICANTS)
        SET @ApplicantId1 = (SELECT distinct(ApplicantId) FROM
MIN_VALUE_APPLICANTS WHERE MINValue = @MIN1)
END
ELSE
BEGIN
        SET @Counter1 = @Counter + 1
        WHILE @Counter1 <= @Length
        BEGIN
                SET @JobId2 = (SELECT JobId FROM PRIORITY WHERE Counter =
@Counter1)
                SET @PlaceCode2 = (SELECT PlaceCode FROM PRIORITY WHERE
Counter = @Counter1)
                print @JobId2
                print @PlaceCode2
                SET @MAX1 = (SELECT max(HValue) FROM H WHERE JobId =
@JobId2 AND PlaceCode = @PlaceCode2 AND ApplicantId IN (SELECT ApplicantId
FROM SAME_MAX_VALUE))
                SET @MultipleMaxValues = (SELECT count(HValue) FROM H
WHERE JobId = @JobId2 AND PlaceCode = @PlaceCode2 AND HValue = @MAX1
AND ApplicantId IN (SELECT ApplicantId FROM SAME_MAX_VALUE))
                PRINT '@MultipleMaxValues = '
                PRINT @MultipleMaxValues
                IF @MultipleMaxValues = 1
                BEGIN
                        SET @ApplicantId2 = (SELECT ApplicantId FROM H WHERE
HValue = @MAX1 AND JobId = @JobId2 AND PlaceCode = @PlaceCode2 AND
ApplicantId IN (SELECT ApplicantId FROM SAME_MAX_VALUE))
                        INSERT INTO ONE_MAX_VALUE
                        VALUES (@JobId2, @PlaceCode2, @ApplicantId2, @Counter1)
                END
                ELSE
                BEGIN
                        DECLARE SameMaxValueCursor1 CURSOR FOR
                        SELECT ApplicantId
                        FROM SAME_MAX_VALUE

                        OPEN SameMaxValueCursor1
                        FETCH NEXT FROM SameMaxValueCursor1
                        INTO @ApplicantId
                        WHILE @@FETCH_STATUS <> -1
```

```
                        BEGIN
                                IF @@FETCH_STATUS <> -2
                                BEGIN
                                        SET @HValue1 = (SELECT HValue FROM H
WHERE JobId = @JobId2 AND PlaceCode = @PlaceCode2 AND ApplicantId =
@ApplicantId)
                                        IF @HValue1 = @MAX1
                                        BEGIN
                                                INSERT INTO
MULTIPLE_MAX_VALUES
                                                VALUES (@JobId2, @PlaceCode2,
@ApplicantId, @Counter1)
                                        END
                                END
                                FETCH NEXT FROM SameMaxValueCursor1
                                INTO @ApplicantId
                        END

                        CLOSE SameMaxValueCursor1
                        DEALLOCATE SameMaxValueCursor1
                END
                SET @Counter1 = @Counter1 + 1
        END

        IF @Length > @Counter + 2
        BEGIN
                --SET @MULTIPLE_MAX_VALUES_Length = (SELECT
max(Counter) FROM MULTIPLE_MAX_VALUES WHERE Counter > @Counter + 1)
                --IF @MULTIPLE_MAX_VALUES_Length > @Counter + 1
                --BEGIN
                        SET @JobId1 = (SELECT JobId FROM PRIORITY WHERE
Counter = @Counter + 1)
                        SET @PlaceCode1 = (SELECT PlaceCode FROM PRIORITY
WHERE Counter = @Counter + 1)
                        SET @MIN1 = (SELECT min(HValue) FROM H WHERE JobId
= @JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId IN (SELECT
ApplicantId FROM MULTIPLE_MAX_VALUES))
                        SET @MAX2 = (SELECT max(HValue) FROM H WHERE JobId
= @JobId1 AND PlaceCode = @PlaceCode1 AND HValue IS NOT NULL AND
                                        ApplicantId NOT IN (SELECT ApplicantId FROM
USED_APPLICANTS WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1)
AND
                                        ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS))

                        IF @MIN1 < @MAX2
```

```
                        BEGIN
                                SET @Eurika = 1
                                SET @ApplicantId1 = (SELECT ApplicantId FROM H
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1 AND HValue = @MIN1)
                        END
                        ELSE
                        BEGIN
                                SET @MinCount = @Length
                                SET @C = 0
                                DECLARE MultipleMaxValueCursor CURSOR FOR
                                SELECT ApplicantId
                                FROM MULTIPLE_MAX_VALUES
                                WHERE Counter = @Counter + 1

                                OPEN MultipleMaxValueCursor
                                FETCH NEXT FROM MultipleMaxValueCursor
                                INTO @ApplicantId
                                WHILE @@FETCH_STATUS <> -1
                                BEGIN
                                        IF @@FETCH_STATUS <> -2
                                        BEGIN
                                                SET @Temp1 = (SELECT
count(ApplicantId) FROM MULTIPLE_MAX_VALUES WHERE ApplicantId =
@ApplicantId)
                                                SET @Temp2 = (SELECT
count(ApplicantId) FROM ONE_MAX_VALUE WHERE ApplicantId = @ApplicantId)
                                                SET @Temp = @Temp1 + @Temp2
                                                SET @C1 = (SELECT max(Counter)
FROM MULTIPLE_MAX_VALUES WHERE ApplicantId = @ApplicantId)
                                                SET @C2 = (SELECT max(Counter)
FROM ONE_MAX_VALUE WHERE ApplicantId = @ApplicantId)
                                                IF @C2 > @C1
                                                        SET @C1 = @C2
                                                IF (@Temp <= @MinCount) AND (@C1
>= @C)
                                                BEGIN
                                                        SET @Eurika = 1
                                                        SET @MinCount = @Temp
                                                        SET @C = @C1
                                                        SET @ApplicantId1 = @ApplicantId
                                                END
                                        END
                                        FETCH NEXT FROM MultipleMaxValueCursor
                                        INTO @ApplicantId
                                END
```

```
                                        CLOSE MultipleMaxValueCursor
                                        DEALLOCATE MultipleMaxValueCursor

                              END
                    --END
          END
          ELSE IF @Length = @Counter + 1
          BEGIN
                    IF EXISTS(SELECT 'True' FROM MULTIPLE_MAX_VALUES
WHERE Counter = @Counter)
                    BEGIN
                              SET @Eurika = 1
                              SET @JobId1 = (SELECT JobId FROM PRIORITY WHERE
Counter = @Counter+ 1)
                              SET @PlaceCode1 = (SELECT PlaceCode FROM PRIORITY
WHERE Counter = @Counter + 1)
                              SET @MIN1 = (SELECT min(HValue) FROM H WHERE JobId
= @JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId IN (SELECT
ApplicantId FROM MULTIPLE_MAX_VALUES WHERE Counter = @Counter))

                              DECLARE MultipleMaxValueCursor1 CURSOR FOR
                              SELECT ApplicantId, Counter
                              FROM MULTIPLE_MAX_VALUES

                              OPEN MultipleMaxValueCursor1
                              FETCH NEXT FROM MultipleMaxValueCursor1
                              INTO @ApplicantId, @Counter2
                              WHILE @@FETCH_STATUS <> -1
                              BEGIN
                                        IF @@FETCH_STATUS <> -2
                                        BEGIN
                                                  SET @HValue1 = (SELECT HValue FROM H
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId =
@ApplicantId)
                                                  IF @HValue1 = @MIN1
                                                            SET @ApplicantId1 = @ApplicantId
                                        END
                                        FETCH NEXT FROM MultipleMaxValueCursor1
                                        INTO @ApplicantId, @Counter2
                              END

                              CLOSE MultipleMaxValueCursor1
                              DEALLOCATE MultipleMaxValueCursor1
                    END
          END
          ELSE IF @Length = @Counter + 2
```

```
        BEGIN
                IF EXISTS(SELECT 'True' FROM MULTIPLE_MAX_VALUES
WHERE Counter = @Counter + 1)
                BEGIN
                        SET @Eurika = 1
                        SET @JobId1 = (SELECT JobId FROM PRIORITY WHERE
Counter = @Counter+ 2)
                        SET @PlaceCode1 = (SELECT PlaceCode FROM PRIORITY
WHERE Counter = @Counter + 2)
                        SET @MIN1 = (SELECT min(HValue) FROM H WHERE JobId
= @JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId IN (SELECT
ApplicantId FROM MULTIPLE_MAX_VALUES WHERE Counter = @Counter + 1))

                        DECLARE MultipleMaxValueCursor1 CURSOR FOR
                        SELECT ApplicantId, Counter
                        FROM MULTIPLE_MAX_VALUES

                        OPEN MultipleMaxValueCursor1
                        FETCH NEXT FROM MultipleMaxValueCursor1
                        INTO @ApplicantId, @Counter2
                        WHILE @@FETCH_STATUS <> -1
                        BEGIN
                                IF @@FETCH_STATUS <> -2
                                BEGIN
                                        SET @HValue1 = (SELECT HValue FROM H
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1 AND ApplicantId =
@ApplicantId)
                                        IF @HValue1 = @MIN1
                                                SET @ApplicantId1 = @ApplicantId
                                END
                                FETCH NEXT FROM MultipleMaxValueCursor1
                                INTO @ApplicantId, @Counter2
                        END

                        CLOSE MultipleMaxValueCursor1
                        DEALLOCATE MultipleMaxValueCursor1
                END
        END

        IF @Eurika = 0
        BEGIN
                SET @ONE_MAX_VALUE_Length = (SELECT max(Counter) FROM
ONE_MAX_VALUE WHERE Counter > @Counter + 1)
                IF @ONE_MAX_VALUE_Length > @Counter + 1
                BEGIN
                        SET @Eurika = 1
```

```
                    SET @Spot = (SELECT max(Counter) FROM
ONE_MAX_VALUE)
                    SET @ApplicantId1 = (SELECT ApplicantId FROM
ONE_MAX_VALUE WHERE Counter = @Spot)
            END
      END

END

IF @Eurika = 0
BEGIN
      DECLARE HCursor1 CURSOR FOR
      SELECT ApplicantId
      FROM H
      WHERE JobId = @JobId AND PlaceCode = @PlaceCode AND HValue =
@MAXValue AND
             ApplicantId IN (SELECT ApplicantId FROM SAME_MAX_VALUE)

      OPEN HCursor1
      FETCH NEXT FROM HCursor1
      INTO @ApplicantId
      WHILE @@FETCH_STATUS <> -1
      BEGIN
            IF @@FETCH_STATUS <> -2
            SET @ApplicantId1 = @ApplicantId
            BREAK
            FETCH NEXT FROM HCursor
            INTO @ApplicantId
      END

      CLOSE HCursor1
      DEALLOCATE HCursor1
END

RETURN @ApplicantId1
GO
```

Name: dec_ H_Fill

```
CREATE PROCEDURE ksergis.dec_H_Fill
AS

DELETE FROM H

INSERT INTO H
SELECT JobId, ApplicantId, PlaceCode, NULL
```

```
FROM JOB_PLACE, APPLICANT

DECLARE @JobId char(10)
DECLARE @ApplicantId char(10)
DECLARE @PlaceCode char(10)
DECLARE @HValue float

DECLARE @Rank int
DECLARE @Specialty int
DECLARE @Qualifications int

DECLARE HCursor CURSOR FOR
SELECT JobId, ApplicantId, PlaceCode
FROM H

OPEN HCursor
FETCH NEXT FROM HCursor
INTO @JobId, @ApplicantId, @PlaceCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @Rank = ksergis.dec_Rank @JobId, @ApplicantId
		EXEC @Specialty = ksergis.dec_Specialty @JobId, @ApplicantId
		EXEC @Qualifications = ksergis.dec_Qualifications @JobId,
@ApplicantId

		IF @Rank = 1 AND @Specialty =1 AND @Qualifications = 1
		BEGIN
			EXEC @HValue = ksergis.dec_H_Function @JobId,
@ApplicantId, @PlaceCode
			UPDATE H
			SET HValue = @HValue
			WHERE JobId = @JobId AND ApplicantId = @ApplicantId AND
PlaceCode= @PlaceCode
		END
	END
	FETCH NEXT FROM HCursor
	INTO @JobId, @ApplicantId, @PlaceCode
END

CLOSE HCursor
DEALLOCATE HCursor
GO
```

Name: dec_ H_ Function
CREATE PROCEDURE ksergis.dec_H_Function (@JobId char(10), @ApplicantId char(10), @PlaceCode char (10)) AS DECLARE @PreferenceCommand int DECLARE @PreferenceApplicant int DECLARE @Language float DECLARE @Credentials float DECLARE @Experience float DECLARE @H float DECLARE @PreferenceCommandCo int DECLARE @PreferenceApplicantCo int DECLARE @LanguageCo int DECLARE @CredentialsCo int DECLARE @ExperienceCo int EXEC @PreferenceCommand = ksergis.dec_PreferenceCommandReturn @JobId, @ApplicantId, @PlaceCode EXEC @PreferenceApplicant = ksergis.dec_PreferenceApplicantReturn @JobId, @ApplicantId, @PlaceCode EXEC @Language = ksergis.dec_Language @JobId, @ApplicantId EXEC @Credentials = ksergis.dec_Credentials @JobId, @ApplicantId EXEC @Experience = ksergis.dec_Experience @JobId, @ApplicantId SET @PreferenceCommandCo = (SELECT CoefficientValue FROM COEFFICIENT WHERE CoefficientId = 'CommandPreferenceCo') SET @PreferenceApplicantCo = (SELECT CoefficientValue FROM COEFFICIENT WHERE CoefficientId = 'ApplicantPreferenceCo') SET @LanguageCo = (SELECT CoefficientValue FROM COEFFICIENT WHERE CoefficientId = 'LanguageCo') SET @CredentialsCo = (SELECT CoefficientValue FROM COEFFICIENT WHERE CoefficientId = 'CredentialsCo') SET @ExperienceCo = (SELECT CoefficientValue FROM COEFFICIENT WHERE CoefficientId = 'ExperienceCo') SET @H = (@PreferenceCommandCo * @PreferenceCommand) + (@PreferenceApplicantCo * @PreferenceApplicant) + (@LanguageCo * @Language) + (@CredentialsCo * @Credentials) + (@ExperienceCo * @Experience) + 1 RETURN @H GO

Name: dec_ H_ Normalize
CREATE PROCEDURE ksergis.dec_H_Normalize

```
AS

EXEC ksergis.dec_MAX_VALUE_ALL_JOBS_Fill

DECLARE @JobId char(10)
DECLARE @ApplicantId char(10)
DECLARE @PlaceCode char(10)
DECLARE @HValue float
DECLARE @MaxValue float

DECLARE HCursor CURSOR FOR
SELECT JobId, ApplicantId, PlaceCode, HValue
FROM H

OPEN HCursor
FETCH NEXT FROM HCursor
INTO @JobId, @ApplicantId, @PlaceCode, @HValue
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		IF @HValue IS NOT NULL
		BEGIN
			SET @MaxValue = (SELECT MAXValue FROM
MAX_VALUE_ALL_JOBS WHERE JobId = @JobId AND PlaceCode = @PlaceCode)
			UPDATE H
			SET HValue = (@HValue * 9 / @MaxValue) + 1
			WHERE JobId = @JobId AND ApplicantId = @ApplicantId AND
PlaceCode= @PlaceCode
		END
	END
	FETCH NEXT FROM HCursor
	INTO @JobId, @ApplicantId, @PlaceCode, @HValue
END

CLOSE HCursor
DEALLOCATE HCursor
GO
```

Name: dec_ Language

```
CREATE PROCEDURE ksergis.dec_Language (@JobId char(10), @ApplicantId
char(10))
AS
DECLARE @LanguageDegree1 float
DECLARE @LanguageDegree2 float
```

```
DECLARE @LanguageCode char(10)
DECLARE @SUM1 float
DECLARE @SUM2 float
DECLARE @ANS float
DECLARE @Count int

SET @SUM1 = 0
SET @SUM2 = 0
SET @Count = 0

DECLARE LanguageCursor CURSOR FOR
SELECT JobId, LanguageCode
FROM JOB_LANGUAGE
WHERE JobId = @JobId

OPEN LanguageCursor
FETCH NEXT FROM LanguageCursor
INTO @JobId, @LanguageCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @LanguageDegree1 = ksergis.dec_Language1 @ApplicantId,
@LanguageCode
		EXEC @LanguageDegree2 = ksergis.dec_Language2 @JobId,
@LanguageCode
		SET @SUM1 = @SUM1 +@LanguageDegree1
		SET @SUM2 = @SUM2 +@LanguageDegree2
		SET @Count = @Count +1
	END
	FETCH NEXT FROM LanguageCursor
	INTO @JobId, @LanguageCode
END

CLOSE LanguageCursor
DEALLOCATE LanguageCursor

IF @SUM1 < @SUM2
	SET @ANS = 0
ELSE
BEGIN
	IF @Count * 200 = @SUM2
		SET @ANS = 1
	ELSE
		SET @ANS = ((@SUM1 - @SUM2) * 9 / ((@Count * 200) - @SUM2))
+ 1
```

```
END

RETURN @ANS
GO
```

Name: dec_ Language1

```
CREATE PROCEDURE ksergis.dec_Language1 (@ApplicantId char(10), @LanguageCode char(10))
AS
DECLARE @LanguageDegree float

IF EXISTS (SELECT LanguageDegree FROM APPLICANT_LANGUAGE WHERE ApplicantId = @ApplicantId AND LanguageCode = @LanguageCode)
	SET @LanguageDegree = (SELECT LanguageDegree FROM APPLICANT_LANGUAGE WHERE ApplicantId = @ApplicantId AND LanguageCode = @LanguageCode)
ELSE
	SET @LanguageDegree = 0
RETURN @LanguageDegree
GO
```

Name: dec_ Language2

```
CREATE PROCEDURE ksergis.dec_Language2 (@JobId char(10), @LanguageCode char(10))
AS
DECLARE @LanguageDegree float
SET @LanguageDegree = (SELECT LanguageDegree FROM JOB_LANGUAGE WHERE JobId = @JobId AND LanguageCode = @LanguageCode)
RETURN @LanguageDegree
GO
```

Name: dec_ Main

```
CREATE PROCEDURE ksergis.dec_Main AS

DELETE FROM DELETED_JOBS
DELETE FROM USED_APPLICANTS
DELETE FROM ASSIGNED_APPLICANTS

EXEC ksergis.dec_H_Fill
EXEC ksergis.dec_H_Normalize
EXEC ksergis.dec_PRIORITY_Fill
EXEC ksergis.dec_COUNTER_Fill
EXEC ksergis.dec_MAX_VALUE_Fill
```

```
EXEC ksergis.dec_DeleteEmptyJobs

DECLARE @Length int
DECLARE @Count int
DECLARE @PriorCount int
DECLARE @Flag bit
DECLARE @CheckHValueExists int

EXEC @Length = ksergis.dec_CountPriorityRecords

SET @Count = 1

WHILE @Count <= @Length
BEGIN
	PRINT @Count
	EXEC @CheckHValueExists = ksergis.dec_CheckHValueExists @Count

	IF @Count = 1
	BEGIN
		SET @Flag = (SELECT Flag FROM PRIORITY WHERE Counter =
@Count)
		IF @CheckHValueExists = 0 AND @Flag = 1
		BEGIN
			EXEC ksergis.dec_DeleteJob
			EXEC @Length = ksergis.dec_CountPriorityRecords
			EXEC ksergis.dec_DeleteJobUsedValues @Count
			EXEC @CheckHValueExists = ksergis.dec_CheckHValueExists
@Count
		END
	END

	IF @CheckHValueExists = 1
	BEGIN
		EXEC ksergis.dec_ComputeMaxValue @Count
		UPDATE PRIORITY
		SET Flag = 1
		WHERE Counter = @Count
		SET @Count = @Count + 1
	END
	ELSE
	BEGIN
		SET @PriorCount = @Count - 1
		EXEC ksergis.dec_SetMAXValueNull @PriorCount
		EXEC ksergis.dec_DeleteJobUsedValues @Count
		SET @Count = @PriorCount
	END
```

```
END

EXEC ksergis.dec_UNASSIGNED_APPLICANTS_Fill
EXEC ksergis.dec_MANIPULATE_SOLUTION_Fill
EXEC ksergis.dec_UNASSIGNED_APPLICANTS_MANIPULATE_Fill
EXEC ksergis.dec_DELETED_JOBS_MANIPULATE_Fill
EXEC ksergis.dec_EstimateFunction
GO
```

Name: dec_ MANIPULATE_SOLUTION_Fill

```
CREATE PROCEDURE ksergis.dec_MANIPULATE_SOLUTION_Fill AS
DELETE FROM MANIPULATE_SOLUTION

INSERT INTO MANIPULATE_SOLUTION
SELECT JobId, PlaceCode, ApplicantId, MAXValue
FROM MAX_VALUE
GO
```

Name: dec_ MANIPULATE_SOLUTION_ InsertRecord

```
CREATE PROCEDURE ksergis.dec_MANIPULATE_SOLUTION_InsertRecord
(@JobId char(10),  @PlaceCode char(10), @ApplicantId  char(10)) AS
DECLARE @HValue float

SET @HValue = (SELECT HValue FROM H WHERE JobId = @JobId AND PlaceCode
= @PlaceCode AND ApplicantId = @ApplicantId)

INSERT INTO MANIPULATE_SOLUTION
VALUES (@JobId, @PlaceCode, @ApplicantId, @HValue)
GO
```

Name: dec_MAX_VALUE_ALL_JOBS_Fill

```
CREATE PROCEDURE ksergis.dec_MAX_VALUE_ALL_JOBS_Fill
AS

DELETE FROM MAX_VALUE_ALL_JOBS

INSERT INTO MAX_VALUE_ALL_JOBS
SELECT JobId, PlaceCode, NULL
FROM JOB_PLACE

DECLARE @JobId char(10)
DECLARE @PlaceCode char(10)
DECLARE @MValue float
```

```
DECLARE MAX_VALUE_ALL_JOBS_Cursor CURSOR FOR
SELECT JobId, PlaceCode
FROM MAX_VALUE_ALL_JOBS

OPEN MAX_VALUE_ALL_JOBS_Cursor
FETCH NEXT FROM MAX_VALUE_ALL_JOBS_Cursor
INTO @JobId, @PlaceCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @MValue = (SELECT MAX(HValue) FROM H WHERE  JobId =
@JobId AND PlaceCode= @PlaceCode)
		UPDATE MAX_VALUE_ALL_JOBS
		SET MaxValue = @MValue
		WHERE JobId = @JobId AND PlaceCode= @PlaceCode
	END
	FETCH NEXT FROM MAX_VALUE_ALL_JOBS_Cursor
	INTO @JobId, @PlaceCode
END

CLOSE MAX_VALUE_ALL_JOBS_Cursor
DEALLOCATE MAX_VALUE_ALL_JOBS_Cursor
GO
```

Name: dec_MAX_VALUE_Fill

```
CREATE PROCEDURE ksergis.dec_MAX_VALUE_Fill
AS

DELETE FROM MAX_VALUE

INSERT INTO MAX_VALUE
SELECT JobId, PlaceCode, NULL, NULL
FROM PRIORITY
GO
```

Name: dec_ PreferenceApplicantReturn

```
CREATE PROCEDURE ksergis.dec_PreferenceApplicantReturn (@JobId char (10),
@ApplicantId char(10), @PlaceCode char(10))
AS
DECLARE @PreferenceApplicant int
```

```
IF EXISTS(SELECT PreferenceApplicant FROM APPLICANT_PREFERENCE
WHERE JobId = @JobId AND ApplicantId = @ApplicantId AND PlaceCode =
@PlaceCode)
BEGIN
	SET @PreferenceApplicant = (SELECT PreferenceApplicant FROM
APPLICANT_PREFERENCE WHERE JobId = @JobId AND ApplicantId =
@ApplicantId AND PlaceCode = @PlaceCode)
	IF @PreferenceApplicant IS NOT NULL
		RETURN 11 - @PreferenceApplicant
	ELSE
		RETURN 0
END
ELSE
	RETURN 0
GO
```

Name: dec_ PreferenceCommandReturn

```
CREATE PROCEDURE ksergis.dec_PreferenceCommandReturn (@JobId char (10),
@ApplicantId char(10), @PlaceCode char(10))
AS
DECLARE @Ans int
IF EXISTS(SELECT PreferenceCommand FROM COMMAND_PREFERENCE
WHERE JobId = @JobId AND ApplicantId = @ApplicantId AND PlaceCode =
@PlaceCode)
BEGIN
	SET @Ans = (SELECT PreferenceCommand FROM
COMMAND_PREFERENCE WHERE JobId = @JobId AND ApplicantId =
@ApplicantId AND PlaceCode = @PlaceCode)
	IF @Ans IS NOT NULL
		RETURN 11 - @Ans
	ELSE
		RETURN 0
END
ELSE
	RETURN 0
GO
```

Name: dec_ PRIORITY_Fill

```
CREATE PROCEDURE ksergis.dec_PRIORITY_Fill
AS

DELETE FROM PRIORITY

INSERT INTO PRIORITY
```

```
SELECT JOB_PLACE.JobId, PlaceCode, Priority, NULL, '0'
FROM JOB_PLACE, JOB
WHERE JOB_PLACE.JobId = JOB.JobId

DECLARE @JobId char(10)
DECLARE @PlaceCode char(10)
DECLARE @Priority int
DECLARE @Priority1 int
DECLARE @Counter int
DECLARE @Counter1 int

SET @Counter1 = 1
SET @Priority1 = 10

WHILE @Priority1 > 0
BEGIN

DECLARE PriorityCursor CURSOR FOR
SELECT JobId, PlaceCode, Priority, Counter
FROM PRIORITY

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Priority, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		IF @Priority1 = (SELECT Priority FROM PRIORITY WHERE JobId =
@JobId AND PlaceCode = @PlaceCode)
		BEGIN
			UPDATE PRIORITY
			SET Counter = @Counter1
			WHERE JobId = @JobId AND PlaceCode = @PlaceCode AND
Priority= @Priority1
			SET @Counter1 = @Counter1 + 1
		END
	END
	FETCH NEXT FROM PriorityCursor
	INTO @JobId, @PlaceCode, @Priority, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor

SET @Priority1 = @Priority1 - 1
```

```
END
GO
```

Name: dec_ QualificationExists1

```
CREATE PROCEDURE ksergis.dec_QualificationExists1 (@ApplicantId char(10),
@QualificationCode char(10))
AS
IF EXISTS(SELECT 'True' FROM QUALIFICATION_APPLICANT WHERE
ApplicantId = @ApplicantId AND QualificationCode = @QualificationCode)
	RETURN 1
ELSE
	RETURN 0
GO
```

Name: dec_ QualificationExists2

```
CREATE PROCEDURE ksergis.dec_QualificationExists2 (@JobId char(10),
@QualificationCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_QUALIFICATION WHERE JobId = @JobId
AND QualificationCode = @QualificationCode)
	RETURN 1
ELSE
	RETURN 0
GO
```

Name: dec_ Qualifications

```
CREATE PROCEDURE ksergis.dec_Qualifications (@JobId char(10), @ApplicantId
char(10))
AS
DECLARE @QualificationCode char(10)
DECLARE @QualificationResult1 int
DECLARE @QualificationResult2 int
DECLARE @Ans int

SET @Ans = 0

DECLARE QualificationsCursor CURSOR FOR
SELECT JobId, QualificationCode
FROM JOB_QUALIFICATION
WHERE JobId = @JobId

OPEN QualificationsCursor
FETCH NEXT FROM QualificationsCursor
```

```
INTO @JobId, @QualificationCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @QualificationResult1 = ksergis.dec_QualificationExists1
@ApplicantId, @QualificationCode
		EXEC @QualificationResult2 = ksergis.dec_QualificationExists2
@JobId, @QualificationCode
		IF @QualificationResult1 = @QualificationResult2 AND
@QualificationResult1 <> 0
			SET @Ans = 1
	END
	FETCH NEXT FROM QualificationsCursor
	INTO @JobId, @QualificationCode
END

CLOSE QualificationsCursor
DEALLOCATE QualificationsCursor

RETURN @Ans
GO
```

Name: dec_ Rank

```
CREATE PROCEDURE ksergis.dec_Rank (@JobId char(10), @ApplicantId char(10))
AS
DECLARE @RankCode char(10)
DECLARE @RankResult1 int
DECLARE @RankResult2 int
DECLARE @Ans int

SET @Ans = 0

DECLARE RankCursor CURSOR FOR
SELECT JobId, RankCode
FROM JOB_RANK
WHERE JobId = @JobId

OPEN RankCursor
FETCH NEXT FROM RankCursor
INTO @JobId, @RankCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
```

```
		EXEC @RankResult1 = ksergis.dec_RankExists1 @ApplicantId,
@RankCode
		EXEC @RankResult2 = ksergis.dec_RankExists2 @JobId, @RankCode
		IF @RankResult1 = @RankResult2 AND @RankResult1 <> 0
			SET @Ans = 1
	END
	FETCH NEXT FROM RankCursor
	INTO @JobId, @RankCode
END

CLOSE RankCursor
DEALLOCATE RankCursor

RETURN @Ans
GO
```

Name: dec_ RankExists1

```
CREATE PROCEDURE ksergis.dec_RankExists1 (@ApplicantId char (10),
@RankCode char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT WHERE ApplicantId = @ApplicantId
AND RankCode = @RankCode)
	RETURN 1
ELSE
	RETURN 0
GO
```

Name: dec_ RankExists2

```
CREATE PROCEDURE ksergis.dec_RankExists2 (@JobId char (10), @RankCode
char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_RANK WHERE JobId = @JobId AND
RankCode = @RankCode)
	RETURN 1
ELSE
	RETURN 0
GO
```

Name: dec_ SetMAXValueNull

```
CREATE PROCEDURE ksergis.dec_SetMAXValueNull (@Counter int)
AS

DECLARE @ApplicantId1 char(10)
```

```
DECLARE @JobId char(10)
DECLARE @JobId1 char(10)
DECLARE @PlaceCode char(10)
DECLARE @PlaceCode1 char(10)

DECLARE PriorityCursor CURSOR FOR
SELECT JobId, PlaceCode, Counter
FROM PRIORITY
WHERE Counter = @Counter

OPEN PriorityCursor
FETCH NEXT FROM PriorityCursor
INTO @JobId, @PlaceCode, @Counter
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		SET @JobId1 = @JobId
		SET @PlaceCode1 = @PlaceCode
	END
	FETCH NEXT FROM PriorityCursor
	INTO @JobId, @PlaceCode, @Counter
END

CLOSE PriorityCursor
DEALLOCATE PriorityCursor

SET @ApplicantId1 = (SELECT ApplicantId FROM MAX_VALUE WHERE JobId =
@JobId1 AND PlaceCode = @PlaceCode1)

PRINT @ApplicantId1
PRINT @JobId1
PRINT @PlaceCode1

DELETE FROM ASSIGNED_APPLICANTS
WHERE ApplicantId = @ApplicantId1

INSERT INTO USED_APPLICANTS
VALUES (@JobId1, @PlaceCode1, @ApplicantId1)

UPDATE MAX_VALUE
SET ApplicantId = NULL, MAXValue = NULL
WHERE JobId = @JobId1 AND PlaceCode = @PlaceCode1
GO
```

Name: dec_ ShowDeletedJobs
CREATE PROCEDURE ksergis.dec_ShowDeletedJobs AS SELECT JobName, PlaceName FROM DELETED_JOBS, JOB, PLACE WHERE DELETED_JOBS.JobId = JOB.JobId AND DELETED_JOBS.PlaceCode = PLACE.PlaceCode GO

Name: dec_ ShowDeletedJobsManipulate
CREATE PROCEDURE ksergis.dec_ShowDeletedJobsManipulate AS SELECT JOB.JobId, JobName, PLACE.PlaceCode, PlaceName FROM DELETED_JOBS_MANIPULATE, JOB, PLACE WHERE DELETED_JOBS_MANIPULATE.JobId = JOB.JobId AND DELETED_JOBS_MANIPULATE.PlaceCode = PLACE.PlaceCode GO

Name: dec_ ShowEstimateFunctionResult
CREATE PROCEDURE ksergis.dec_ShowEstimateFunctionResult AS SELECT Result FROM ESTIMATE_FUNCTION_RESULT GO

Name: dec_ ShowJobNameOnJobId
CREATE PROCEDURE ksergis.dec_ShowJobNameOnJobId (@JobId char(10)) AS SELECT JobName FROM JOB WHERE JobId = @JobId GO

Name: dec_ ShowManipulateSolution
CREATE PROCEDURE ksergis.dec_ShowManipulateSolution AS SELECT JOB.JobId, JobName, PLACE.PlaceCode, PlaceName, APPLICANT.ApplicantId, FirstName, LastName, MAXValue FROM MANIPULATE_SOLUTION, JOB, PLACE, APPLICANT WHERE MANIPULATE_SOLUTION.JobId = JOB.JobId AND MANIPULATE_SOLUTION.PlaceCode = PLACE.PlaceCode AND MANIPULATE_SOLUTION.ApplicantId = APPLICANT.ApplicantId GO

Name: dec_ ShowNotNullHValue
CREATE PROCEDURE ksergis.dec_ShowNotNullHValue AS

```
SELECT *
FROM H
WHERE HValue IS NOT NULL
GO
```

Name: dec_ ShowPlaceNameOnPlaceCode

```
CREATE PROCEDURE ksergis.dec_ShowPlaceNameOnPlaceCode (@PlaceCode char(10)) AS
SELECT PlaceName
FROM PLACE
WHERE PlaceCode = @PlaceCode
GO
```

Name: dec_ ShowSolution

```
CREATE PROCEDURE ksergis.dec_ShowSolution AS
SELECT JobName, PlaceName, APPLICANT.ApplicantId, FirstName, LastName, MAXValue
FROM MAX_VALUE, JOB, PLACE, APPLICANT
WHERE MAX_VALUE.JobId = JOB.JobId AND MAX_VALUE.PlaceCode = PLACE.PlaceCode AND
	MAX_VALUE.ApplicantId = APPLICANT.ApplicantId
GO
```

Name: dec_ ShowUnassignedApplicants

```
CREATE PROCEDURE ksergis.dec_ShowUnassignedApplicants AS

SELECT APPLICANT.ApplicantId, FirstName, LastName
FROM APPLICANT, UNASSIGNED_APPLICANTS
WHERE APPLICANT.ApplicantId = UNASSIGNED_APPLICANTS.ApplicantId
GO
```

Name: dec_ ShowUnassignedApplicantsManipulate

```
CREATE PROCEDURE ksergis.dec_ShowUnassignedApplicantsManipulate AS

SELECT APPLICANT.ApplicantId, FirstName, LastName
FROM APPLICANT, UNASSIGNED_APPLICANTS_MANIPULATE
WHERE APPLICANT.ApplicantId = UNASSIGNED_APPLICANTS_MANIPULATE.ApplicantId
GO
```

Name: dec_ Specialty

```
CREATE PROCEDURE ksergis.dec_Specialty (@JobId char(10), @ApplicantId char(10))
AS
DECLARE @SpecialtyCode char(10)
DECLARE @SpecialtyResult1 int
DECLARE @SpecialtyResult2 int
DECLARE @Ans int

SET @Ans = 0

DECLARE SpecialtyCursor CURSOR FOR
SELECT JobId, SpecialtyCode
FROM JOB_SPECIALTY
WHERE JobId = @JobId

OPEN SpecialtyCursor
FETCH NEXT FROM SpecialtyCursor
INTO @JobId, @SpecialtyCode
WHILE @@FETCH_STATUS <> -1
BEGIN
	IF @@FETCH_STATUS <> -2
	BEGIN
		EXEC @SpecialtyResult1 = ksergis.dec_SpecialtyExists1 @ApplicantId, @SpecialtyCode
		EXEC @SpecialtyResult2 = ksergis.dec_SpecialtyExists2 @JobId, @SpecialtyCode
		IF @SpecialtyResult1 = @SpecialtyResult2 AND @SpecialtyResult1 <> 0
			SET @Ans = 1
	END
	FETCH NEXT FROM SpecialtyCursor
	INTO @JobId, @SpecialtyCode
END

CLOSE SpecialtyCursor
DEALLOCATE SpecialtyCursor

RETURN @Ans
GO
```

Name: dec_ SpecialtyExists1

```
CREATE PROCEDURE ksergis.dec_SpecialtyExists1 (@ApplicantId char (10), @SpecialtyCode char(10))
AS
IF EXISTS(SELECT 'True' FROM APPLICANT WHERE ApplicantId = @ApplicantId
```

```
AND SpecialtyCode = @SpecialtyCode)
        RETURN 1
ELSE
        RETURN 0
GO
```

Name: dec_ SpecialtyExists2

```
CREATE PROCEDURE ksergis.dec_SpecialtyExists2 (@JobId char (10),
@SpecialtyCode char(10))
AS
IF EXISTS(SELECT 'True' FROM JOB_SPECIALTY WHERE JobId = @JobId AND
SpecialtyCode = @SpecialtyCode)
        RETURN 1
ELSE
        RETURN 0
GO
```

Name: dec_UNASSIGNED_APPLICANTS_Fill

```
CREATE PROCEDURE ksergis.dec_UNASSIGNED_APPLICANTS_Fill AS
DELETE FROM UNASSIGNED_APPLICANTS

INSERT INTO UNASSIGNED_APPLICANTS
SELECT ApplicantId
FROM APPLICANT
WHERE ApplicantId NOT IN (SELECT ApplicantId FROM
ASSIGNED_APPLICANTS)
GO
```

Name: dec_UNASSIGNED_APPLICANTS_MANIPULATE_DeleteRecord

```
CREATE PROCEDURE
ksergis.dec_UNASSIGNED_APPLICANTS_MANIPULATE_DeleteRecord
(@ApplicantId char(10)) AS
DELETE FROM UNASSIGNED_APPLICANTS_MANIPULATE
WHERE ApplicantId = @ApplicantId
GO
```

Name: dec_UNASSIGNED_APPLICANTS_MANIPULATE_ Fill

```
CREATE PROCEDURE
ksergis.dec_UNASSIGNED_APPLICANTS_MANIPULATE_Fill AS
DELETE FROM UNASSIGNED_APPLICANTS_MANIPULATE

INSERT INTO UNASSIGNED_APPLICANTS_MANIPULATE
```

```
SELECT *
FROM UNASSIGNED_APPLICANTS
GO
```

Name: DeleteApplicantIdOnApplicantCredentials
CREATE PROCEDURE ksergis.DeleteApplicantIdOnApplicantCredentials (@ApplicantId char(10), @CredentialsId char(10)) AS DELETE FROM APPLICANT_CREDENTIALS WHERE ApplicantId = @ApplicantId AND CredentialsId = @CredentialsId GO

Name: DeleteApplicantIdOnApplicantLanguage
CREATE PROCEDURE ksergis.DeleteApplicantIdOnApplicantLanguage (@ApplicantId char(10), @LanguageCode char(10)) AS DELETE FROM APPLICANT_LANGUAGE WHERE ApplicantId = @ApplicantId AND LanguageCode = @LanguageCode GO

Name: DeleteApplicantIdOnQualificationApplicant
CREATE PROCEDURE ksergis.DeleteApplicantIdOnQualificationApplicant (@ApplicantId char(10), @QualificationCode char(10)) AS DELETE FROM QUALIFICATION_APPLICANT WHERE ApplicantId = @ApplicantId AND QualificationCode = @QualificationCode GO

Name: DeleteApplicantPreference
CREATE PROCEDURE ksergis.DeleteApplicantPreference (@ApplicantId char(10), @PreferenceApplicant char(10)) AS DELETE FROM APPLICANT_PREFERENCE WHERE ApplicantId = @ApplicantId AND PreferenceApplicant = @PreferenceApplicant GO

Name: DeleteApplicants
CREATE PROCEDURE ksergis.DeleteApplicants (@ApplicantId char(10)) AS DELETE FROM APPLICANT

WHERE ApplicantId = @ApplicantId GO

Name: DeleteCoefficient
CREATE PROCEDURE ksergis.DeleteCoefficient (@CoefficientId char(30)) AS DELETE FROM COEFFICIENT WHERE CoefficientId = @CoefficientId GO

Name: DeleteCommandPreference
CREATE PROCEDURE ksergis.DeleteCommandPreference (@PlaceCode char(10), @JobId char(10), @PreferenceCommand char(10), @ApplicantId char(10)) AS DELETE FROM COMMAND_PREFERENCE WHERE PlaceCode = @PlaceCode AND JobId = @JobId AND PreferenceCommand = @PreferenceCommand AND ApplicantId = @ApplicantId GO

Name: DeleteCommands
CREATE PROCEDURE ksergis.DeleteCommands (@CommandCode char(10)) AS DELETE FROM COMMAND WHERE CommandCode = @CommandCode GO

Name: DeleteCredentials
CREATE PROCEDURE ksergis.DeleteCredentials (@CredentialsId char(10)) AS DELETE FROM CREDENTIALS WHERE CredentialsId = @CredentialsId GO

Name: DeleteCredentialsIdOnJobCredentials
CREATE PROCEDURE ksergis.DeleteCredentialsIdOnJobCredentials (@JobId char(10), @CredentialsId char(10)) AS DELETE FROM JOB_CREDENTIALS WHERE JobId = @JobId AND CredentialsId = @CredentialsId GO

Name: DeleteJobs
CREATE PROCEDURE ksergis.DeleteJobs (@JobId char(10)) AS DELETE FROM JOB WHERE JobId = @JobId GO

Name: DeleteLanguageCodeOnJobLanguage
CREATE PROCEDURE ksergis.DeleteLanguageCodeOnJobLanguage (@JobId char(10), @LanguageCode char(10)) AS DELETE FROM JOB_LANGUAGE WHERE JobId = @JobId AND LanguageCode = @LanguageCode GO

Name: DeleteLanguages
CREATE PROCEDURE ksergis.DeleteLanguages (@LanguageCode char(10)) AS DELETE FROM LANGUAGE WHERE LanguageCode = @LanguageCode GO

Name: DeletePlaceCodeOnJobPlace
CREATE PROCEDURE ksergis.DeletePlaceCodeOnJobPlace (@JobId char(10), @PlaceCode char(10)) AS DELETE FROM JOB_PLACE WHERE JobId = @JobId AND PlaceCode = @PlaceCode GO

Name: DeletePlaces
CREATE PROCEDURE ksergis.DeletePlaces (@PlaceCode char(10)) AS DELETE FROM PLACE WHERE PlaceCode = @PlaceCode GO

Name: DeleteQualificationCodeOnJobQualification
CREATE PROCEDURE ksergis.DeleteQualificationCodeOnJobQualification (@JobId char(10), @QualificationCode char(10))

AS DELETE FROM JOB_QUALIFICATION WHERE JobId = @JobId AND QualificationCode = @QualificationCode GO

Name: DeleteQualifications
CREATE PROCEDURE ksergis.DeleteQualifications (@QualificationCode char(10)) AS DELETE FROM QUALIFICATION WHERE QualificationCode = @QualificationCode GO

Name: DeleteRankCodeOnJobRank
CREATE PROCEDURE ksergis.DeleteRankCodeOnJobRank (@JobId char(10), @RankCode char(10)) AS DELETE FROM JOB_RANK WHERE JobId = @JobId AND RankCode = @RankCode GO

Name: DeleteRanks
CREATE PROCEDURE ksergis.DeleteRanks (@RankCode char(10)) AS DELETE FROM RANK WHERE RankCode = @RankCode GO

Name: DeleteSpecialties
CREATE PROCEDURE ksergis.DeleteSpecialties (@SpecialtyCode char(10)) AS DELETE FROM SPECIALTY WHERE SpecialtyCode = @SpecialtyCode GO

Name: DeleteSpecialtyCodeOnJobSpecialty
CREATE PROCEDURE ksergis.DeleteSpecialtyCodeOnJobSpecialty (@JobId char(10), @SpecialtyCode char(10)) AS DELETE FROM JOB_SPECIALTY WHERE JobId = @JobId AND SpecialtyCode = @SpecialtyCode GO

Name: FindPlaceCodeJobId
CREATE PROCEDURE ksergis.FindPlaceCodeJobId (@CommandCode char(10), @JobName char(30), @PlaceName char(50), @PreferenceCommand int) AS SELECT PLACE.PlaceCode, JOB.JobId FROM EXPERIENCE_PREFERENCE, JOB, PLACE WHERE EXPERIENCE_PREFERENCE.CommandCode=@CommandCode AND JobName = @JobName AND PlaceName = @PlaceName AND JOB.JobId = EXPERIENCE_PREFERENCE.JobId AND PLACE.PlaceCode = EXPERIENCE_PREFERENCE.PlaceCode AND PreferenceCommand = @PreferenceCommand GO

Name: InsertCoefficient
CREATE PROCEDURE ksergis.InsertCoefficient (@CoefficientId char(30), @CoefficientValue int) AS INSERT INTO COEFFICIENT VALUES (@CoefficientId, @CoefficientValue) GO

Name: InsertDate
CREATE PROCEDURE ksergis.InsertDate (@ApplicantId char(10), @ReportDate varchar(10), @DetachDate varchar(10)) AS DECLARE @d_ReportDate datetime DECLARE @d_DetachDate datetime SET @d_ReportDate = @ReportDate SET @d_DetachDate = @DetachDate UPDATE ASSIGNMENT SET ReportDate = @d_ReportDate, DetachDate = @DetachDate WHERE ApplicantId = @ApplicantId GO

Name: InsertExperience
CREATE PROCEDURE ksergis.InsertExperience (@JobId char(10), @ApplicantId char(10), @Experience float) AS INSERT INTO EXPERIENCE VALUES (@JobId, @ApplicantId, @Experience) GO

Name: SearchCommandName
CREATE PROCEDURE ksergis.SearchCommandName (@UserName varchar(50)) AS SELECT CommandName, CommandCode FROM COMMAND WHERE UserName=@UserName GO

Name: SearchLastName
CREATE PROCEDURE ksergis.SearchLastName (@UserName varchar(50)) AS SELECT LastName, ApplicantId, DetailerCheck FROM APPLICANT WHERE UserName=@UserName GO

Name: ShowAllAssignmentInfo
CREATE PROCEDURE ksergis.ShowAllAssignmentInfo AS SELECT ASSIGNMENT.JobId, JobName, ASSIGNMENT.PlaceCode, PlaceName, ASSIGNMENT.ApplicantId, FirstName, LastName, ReportDate, DetachDate FROM ASSIGNMENT, JOB, PLACE, APPLICANT WHERE ASSIGNMENT.JobId = JOB.JobId AND ASSIGNMENT.PlaceCode = PLACE.PlaceCode AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId GO

Name: ShowAllJobIdRelatedData
CREATE PROCEDURE ksergis.ShowAllJobIdRelatedData (@JobId char(10)) AS SELECT JOB.JobId, JobName, ExperienceRequired, RankName, LanguageName, LanguageDegree, SpecialtyName, QualificationName, PlaceName, CredentialsName, CredentialsGrade FROM JOB, JOB_RANK, RANK, JOB_LANGUAGE, LANGUAGE, SPECIALTY, JOB_SPECIALTY, QUALIFICATION, JOB_QUALIFICATION, PLACE, JOB_PLACE, CREDENTIALS, JOB_CREDENTIALS WHERE JOB.JobId = @JobId AND JOB.JobId = JOB_RANK.JobId AND JOB_RANK.RankCode = RANK.RankCode AND JOB_LANGUAGE.LanguageCode = LANGUAGE.LanguageCode AND JOB_LANGUAGE.JobId = JOB.JobId AND JOB_SPECIALTY.SpecialtyCode = SPECIALTY.SpecialtyCode AND JOB_SPECIALTY.JobId = JOB.JobId AND JOB_QUALIFICATION.QualificationCode = QUALIFICATION.QualificationCode AND JOB_QUALIFICATION.JobId = JOB.JobId

```
	AND JOB_PLACE.PlaceCode = PLACE.PlaceCode AND JOB_PLACE.JobId = JOB.JobId
	AND JOB_CREDENTIALS.CredentialsId = CREDENTIALS.CredentialsId AND JOB_CREDENTIALS.JobId = JOB.JobId
GO
```

Name: ShowApplicantAddressPhoneData

```
CREATE PROCEDURE ksergis.ShowApplicantAddressPhoneData (@ApplicantId char(10))
AS
SELECT FirstName, LastName, MiddleName, UserName, Password, EmailAddress, CityOrTown, Street, Appartment, ZIP, HomePhoneNumber, CellPhoneNumber, OtherPhoneNumber
FROM APPLICANT, ADDRESS, PHONE
WHERE APPLICANT.ApplicantId = @ApplicantId AND ADDRESS.ApplicantId = @ApplicantId AND  PHONE.ApplicantId = @ApplicantId
GO
```

Name: ShowApplicantData

```
CREATE PROCEDURE ksergis.ShowApplicantData AS
SELECT FirstName, LastName, MiddleName, CityOrTown, Street, Appartment
FROM dbo.APPLICANT, dbo.ADDRESS
WHERE UserName = Request.Form(“UserName”) AND Password = Request.Form(“Password”)
GO
```

Name: ShowApplicantDataOnJobIdFromEXPERIENCE

```
CREATE PROCEDURE ksergis.ShowApplicantDataOnJobIdFromEXPERIENCE (@JobId char(10)) AS
SELECT APPLICANT.ApplicantId, FirstName, LastName
FROM APPLICANT, EXPERIENCE
WHERE APPLICANT.ApplicantId = EXPERIENCE.ApplicantId AND EXPERIENCE.JobId = @JobId
GO
```

Name: ShowApplicantId

```
CREATE PROCEDURE ksergis.ShowApplicantId AS
SELECT ApplicantId
FROM APPLICANT
GO
```

Name: ShowApplicantIdFromUserName

```
CREATE PROCEDURE ksergis.ShowApplicantIdFromUserName (@UserName char(50))
AS
SELECT ApplicantId
FROM dbo.APPLICANT
WHERE UserName = @UserName
GO
```

Name: ShowApplicantIdLastNameFirstName

```
CREATE PROCEDURE ksergis.ShowApplicantIdLastNameFirstName AS
SELECT ApplicantId, FirstName, LastName, RankName
FROM APPLICANT, RANK
WHERE APPLICANT.RankCode = RANK.RankCode
GO
```

Name: ShowApplicantIdLastNameFirstNameOnApplicantId

```
CREATE PROCEDURE ksergis.ShowApplicantIdLastNameFirstNameOnApplicantId (@ApplicantId char(10)) AS
SELECT ApplicantId, FirstName, LastName
FROM APPLICANT
WHERE ApplicantId = @ApplicantId
GO
```

Name: ShowApplicantIdLastNameFirstNameRankNameOnApplicantId

```
CREATE PROCEDURE ksergis.ShowApplicantIdLastNameFirstNameRankNameOnApplicantId (@ApplicantId char(10)) AS
SELECT ApplicantId, FirstName, LastName, RankName
FROM APPLICANT, RANK
WHERE APPLICANT.RankCode = RANK.RankCode AND ApplicantId = @ApplicantId
GO
```

Name: ShowApplicantIdLastNameFirstNameWORank

```
CREATE PROCEDURE ksergis.ShowApplicantIdLastNameFirstNameWORank AS
SELECT ApplicantId, FirstName, LastName
FROM APPLICANT
GO
```

Name: ShowApplicantPreferences

```
CREATE PROCEDURE ksergis.ShowApplicantPreferences (@ApplicantId varchar(10))
AS
SELECT PreferenceApplicant, JOB.JobName, PlaceName
FROM APPLICANT_PREFERENCE, JOB, PLACE
WHERE ApplicantId=@ApplicantId AND JOB.JobId = APPLICANT_PREFERENCE.JobId AND PLACE.PlaceCode = APPLICANT_PREFERENCE.PlaceCode
ORDER BY PreferenceApplicant, PlaceName, JOB.JobName
GO
```

Name: ShowApplicantRankSpecialtySeaTimeForRank

```
CREATE PROCEDURE ksergis.ShowApplicantRankSpecialtySeaTimeForRank (@ApplicantId char (10))
AS
SELECT RankName, SpecialtyName, SeaTimeForRank
FROM APPLICANT, SPECIALTY, RANK
WHERE ApplicantId = @ApplicantId AND APPLICANT.RankCode = RANK.RankCode AND APPLICANT.SpecialtyCode = SPECIALTY.SpecialtyCode
GO
```

Name: ShowCoefficients

```
CREATE PROCEDURE ksergis.ShowCoefficients AS
SELECT *
FROM COEFFICIENT
GO
```

Name: ShowCommandCode

```
CREATE PROCEDURE ksergis.ShowCommandCode AS
SELECT CommandCode, CommandName
FROM COMMAND
GO
```

Name: ShowCommandsData

```
CREATE PROCEDURE ksergis.ShowCommandsData AS
SELECT *
FROM COMMAND
GO
```

Name: ShowCommandsPreferences

```
CREATE PROCEDURE ksergis.ShowCommandsPreferences (@CommandCode char(50))
```

```
AS
SELECT JOB.JobName, PlaceName, PreferenceCommand, LastName, FirstName, RankName
FROM COMMAND_PREFERENCE, JOB, PLACE, APPLICANT, RANK
WHERE COMMAND_PREFERENCE.CommandCode=@CommandCode AND COMMAND_PREFERENCE.ApplicantId = APPLICANT.ApplicantId AND APPLICANT.RankCode = RANK.RankCode AND JOB.JobId = COMMAND_PREFERENCE.JobId AND PLACE.PlaceCode = COMMAND_PREFERENCE.PlaceCode
ORDER BY PlaceName, JOB.JobName, PreferenceCommand, RankName, LastName, FirstName
GO
```

Name: ShowCommandsPreferencesForDelete

```
CREATE PROCEDURE ksergis.ShowCommandsPreferencesForDelete (@CommandCode char(50))
AS
SELECT PlaceName, JOB.JobName, PreferenceCommand, APPLICANT.ApplicantId, LastName, FirstName, RankName, JOB.JobId, PLACE.PlaceCode
FROM COMMAND_PREFERENCE, JOB, PLACE, APPLICANT, RANK
WHERE COMMAND_PREFERENCE.CommandCode=@CommandCode AND COMMAND_PREFERENCE.ApplicantId = APPLICANT.ApplicantId AND APPLICANT.RankCode = RANK.RankCode AND JOB.JobId = COMMAND_PREFERENCE.JobId AND PLACE.PlaceCode = COMMAND_PREFERENCE.PlaceCode
ORDER BY PlaceName, JOB.JobName, PreferenceCommand, RankName, LastName, FirstName
GO
```

Name: ShowCommandsPreferencesOnPlaceCode

```
CREATE PROCEDURE ksergis.ShowCommandsPreferencesOnPlaceCode (@CommandCode char(50), @PlaceCode char(10))
AS
SELECT JOB.JobName, PreferenceCommand, LastName, FirstName, RankName
FROM COMMAND_PREFERENCE, JOB, APPLICANT, RANK
WHERE COMMAND_PREFERENCE.CommandCode=@CommandCode AND COMMAND_PREFERENCE.ApplicantId = APPLICANT.ApplicantId AND APPLICANT.RankCode = RANK.RankCode AND JOB.JobId = COMMAND_PREFERENCE.JobId AND COMMAND_PREFERENCE.PlaceCode = @PlaceCode
ORDER BY JOB.JobName, PreferenceCommand, RankName, LastName, FirstName
GO
```

Name: ShowCredentialsGrade
CREATE PROCEDURE ksergis.ShowCredentialsGrade (@ApplicantId char(10), @CredentialsId char(10))AS SELECT CredentialsGrade FROM APPLICANT_CREDENTIALS WHERE ApplicantId = @ApplicantId AND CredentialsId = @CredentialsId GO

Name: ShowCredentialsId
CREATE PROCEDURE ksergis.ShowCredentialsId AS SELECT CredentialsId, CredentialsName FROM CREDENTIALS GO

Name: ShowCredentialsIdOnApplicantId
CREATE PROCEDURE ksergis.ShowCredentialsIdOnApplicantId (@ApplicantId char(10))AS SELECT CREDENTIALS.CredentialsId, CredentialsName, CredentialsGrade FROM CREDENTIALS, APPLICANT, APPLICANT_CREDENTIALS WHERE APPLICANT.ApplicantId = @ApplicantId AND APPLICANT.ApplicantId = APPLICANT_CREDENTIALS.ApplicantId AND APPLICANT_CREDENTIALS.CredentialsId = CREDENTIALS.CredentialsId GO

Name: ShowCredentialsIdOnJobId
CREATE PROCEDURE ksergis.ShowCredentialsIdOnJobId (@JobId char(10)) AS SELECT distinct(JOB_CREDENTIALS.CredentialsId), CredentialsName FROM CREDENTIALS, JOB_CREDENTIALS WHERE CREDENTIALS.CredentialsId = JOB_CREDENTIALS.CredentialsId AND JobId = @JobId GO

Name: ShowCredentialsNameCredentialsGradeOnJobId
CREATE PROCEDURE ksergis.ShowCredentialsNameCredentialsGradeOnJobId (@JobId char(10)) AS SELECT CredentialsName, CredentialsGrade FROM JOB_CREDENTIALS, CREDENTIALS WHERE JobId = @JobId AND JOB_CREDENTIALS.CredentialsId = CREDENTIALS.CredentialsId

GO

Name: ShowCurrentAssignment
CREATE PROCEDURE ksergis.ShowCurrentAssignment (@ApplicantId char(10)) AS SELECT LastName, FirstName, PlaceName, PlaceImage, JobName, ReportDate, DetachDate FROM ASSIGNMENT, JOB, PLACE, APPLICANT WHERE ASSIGNMENT.ApplicantId = @ApplicantId AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId AND PLACE.PlaceCode = ASSIGNMENT.PlaceCode AND JOB.JobId = ASSIGNMENT.JobId GO

Name: ShowExperienceOnJobIdJobName
CREATE PROCEDURE ksergis.ShowExperienceOnJobIdJobName (@JobId char(10), @ApplicantId char(10))AS SELECT Experience FROM EXPERIENCE WHERE JobId = @JobId AND ApplicantId= @ApplicantId GO

Name: ShowExperiencePerJobOfficer
CREATE PROCEDURE ksergis.ShowExperiencePerJobOfficer AS SELECT EXPERIENCE.JobId, JobName, APPLICANT.ApplicantId, LastName, FirstName, Experience FROM EXPERIENCE, JOB, APPLICANT WHERE EXPERIENCE.JobId = JOB.JobId AND EXPERIENCE.ApplicantId = APPLICANT.ApplicantId GO

Name: ShowExperienceRequired
CREATE PROCEDURE ksergis.ShowExperienceRequired (@JobId char(10), @JobName char(30)) AS SELECT ExperienceRequired FROM Job WHERE JobId = @JobId AND JobName = @JobName GO

Name: ShowJobId

CREATE PROCEDURE ksergis.ShowJobId AS SELECT JobId, JobName FROM JOB GO

Name: ShowJobIdJobNameFromEXPERIENCE
CREATE PROCEDURE ksergis.ShowJobIdJobNameFromEXPERIENCE AS SELECT distinct (JOB.JobId), JobName FROM EXPERIENCE, JOB WHERE EXPERIENCE.JobId = JOB.JobId GO

Name: ShowJobIdOnPlaceCode
CREATE PROCEDURE ksergis.ShowJobIdOnPlaceCode (@PlaceCode char(10)) AS SELECT JOB_PLACE.JobId, JobName FROM JOB_PLACE, JOB WHERE PlaceCode = @PlaceCode AND JOB_PLACE.JobId = JOB.JobId GO

Name: ShowJobIdPlaceCodeApplicantIdFromASSIGNMENT
CREATE PROCEDURE ksergis.ShowJobIdPlaceCodeApplicantIdFromASSIGNMENT AS SELECT ASSIGNMENT.JobId, JobName, ASSIGNMENT.PlaceCode, PlaceName, ASSIGNMENT.ApplicantId, FirstName, LastName FROM ASSIGNMENT, JOB, PLACE, APPLICANT WHERE ASSIGNMENT.JobId = JOB.JobId AND ASSIGNMENT.PlaceCode = PLACE.PlaceCode AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId GO

Name: ShowJobIdPlaceCodeApplicantIdFromASSIGNMENTForUpdate
CREATE PROCEDURE ksergis.ShowJobIdPlaceCodeApplicantIdFromASSIGNMENTForUpdate AS SELECT ASSIGNMENT.JobId, JobName, ASSIGNMENT.PlaceCode, PlaceName, ASSIGNMENT.ApplicantId, FirstName, LastName, ReportDate, DetachDate FROM ASSIGNMENT, JOB, PLACE, APPLICANT WHERE ASSIGNMENT.JobId = JOB.JobId AND ASSIGNMENT.PlaceCode = PLACE.PlaceCode AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId AND (ReportDate IS NOT NULL OR DetachDate IS NOT NULL) GO

Name: ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENT

```
CREATE PROCEDURE
ksergis.ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENT
(@ApplicantId char(10)) AS
SELECT ASSIGNMENT.JobId, JobName, ASSIGNMENT.PlaceCode, PlaceName,
ASSIGNMENT.ApplicantId, FirstName, LastName
FROM ASSIGNMENT, JOB, PLACE, APPLICANT
WHERE ASSIGNMENT.JobId = JOB.JobId AND ASSIGNMENT.PlaceCode =
PLACE.PlaceCode AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId
	AND ASSIGNMENT.ApplicantId = @ApplicantId
GO
```

Name: ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENTForUpdate

```
CREATE PROCEDURE
ksergis.ShowJobIdPlaceCodeApplicantIdOnApplicantIdFromASSIGNMENTForUpdate
(@ApplicantId char(10)) AS
SELECT ASSIGNMENT.JobId, JobName, ASSIGNMENT.PlaceCode, PlaceName,
ASSIGNMENT.ApplicantId, FirstName, LastName, ReportDate, DetachDate
FROM ASSIGNMENT, JOB, PLACE, APPLICANT
WHERE ASSIGNMENT.JobId = JOB.JobId AND ASSIGNMENT.PlaceCode =
PLACE.PlaceCode AND ASSIGNMENT.ApplicantId = APPLICANT.ApplicantId
	AND ASSIGNMENT.ApplicantId = @ApplicantId
GO
```

Name: ShowLanguageCode

```
CREATE PROCEDURE ksergis.ShowLanguageCode AS
SELECT LanguageCode, LanguageName
FROM LANGUAGE
GO
```

Name: ShowLanguageCodeOnApplicantId

```
CREATE PROCEDURE ksergis.ShowLanguageCodeOnApplicantId (@ApplicantId
char(10))AS
SELECT LANGUAGE.LanguageCode, LanguageName, LanguageDegree
FROM LANGUAGE, APPLICANT, APPLICANT_LANGUAGE
WHERE APPLICANT.ApplicantId = @ApplicantId
    AND APPLICANT.ApplicantId = APPLICANT_LANGUAGE.ApplicantId
    AND APPLICANT_LANGUAGE.LanguageCode = LANGUAGE.LanguageCode
GO
```

Name: ShowLanguageCodeOnJobId

```
CREATE PROCEDURE ksergis.ShowLanguageCodeOnJobId (@JobId char(10))
```

AS SELECT LANGUAGE.LanguageCode, LanguageName FROM LANGUAGE, JOB_LANGUAGE WHERE LANGUAGE.LanguageCode = JOB_LANGUAGE.LanguageCode AND JOB_LANGUAGE.JobId = @JobId GO

Name: ShowLanguageDegree
CREATE PROCEDURE ksergis.ShowLanguageDegree (@ApplicantId char(10), @LanguageCode char(10))AS SELECT LanguageDegree FROM APPLICANT_LANGUAGE WHERE ApplicantId = @ApplicantId AND LanguageCode = @LanguageCode GO

Name: ShowLanguageNameLanguageDegreeOnJobId
CREATE PROCEDURE ksergis.ShowLanguageNameLanguageDegreeOnJobId (@JobId char(10)) AS SELECT LanguageName, LanguageDegree FROM JOB_LANGUAGE, LANGUAGE WHERE JobId = @JobId AND JOB_LANGUAGE.LanguageCode = LANGUAGE.LanguageCode GO

Name: ShowPlaceCode
CREATE PROCEDURE ksergis.ShowPlaceCode AS SELECT PlaceCode, PlaceName FROM PLACE GO

Name: ShowPlaceCodeOnCommandCode
CREATE PROCEDURE ksergis.ShowPlaceCodeOnCommandCode (@CommandCode char(10))AS SELECT PlaceCode, PlaceName FROM PLACE WHERE CommandCode = @CommandCode GO

Name: ShowPlaceCodeOnJobId

```
CREATE PROCEDURE ksergis.ShowPlaceCodeOnJobId (@JobId varchar(10))
AS
SELECT JOB_PLACE.PlaceCode, PlaceName
FROM JOB_PLACE, PLACE
WHERE JobId = @JobId AND JOB_PLACE.PlaceCode = PLACE.PlaceCode
GO
```

Name: ShowPlaceData

```
CREATE PROCEDURE ksergis.ShowPlaceData AS
SELECT PlaceImage, PlaceCode, PlaceName, PLACE.CommandCode, CommandName
FROM PLACE, COMMAND
WHERE PLACE.CommandCode = COMMAND.CommandCode
GO
```

Name: ShowPlaceImage

```
CREATE PROCEDURE ksergis.ShowPlaceImage (@CommandCode char(50))
AS
SELECT DISTINCT (PlaceName), PlaceImage,
COMMAND_PREFERENCE.PlaceCode
FROM COMMAND_PREFERENCE, JOB, PLACE
WHERE COMMAND_PREFERENCE.CommandCode=@CommandCode AND
PLACE.PlaceCode = COMMAND_PREFERENCE.PlaceCode
ORDER BY PlaceName
GO
```

Name: ShowPlaceNamePlaceImageCommandNameOnJobId

```
CREATE PROCEDURE ksergis.ShowPlaceNamePlaceImageCommandNameOnJobId
(@JobId char(10))
AS
SELECT PlaceImage, PlaceName, CommandName
FROM JOB_PLACE, PLACE, COMMAND
WHERE JobId = @JobId AND JOB_PLACE.PlaceCode = PLACE.PlaceCode AND
PLACE.CommandCode = COMMAND.CommandCode
GO
```

Name: ShowQualificationCode

```
CREATE PROCEDURE ksergis.ShowQualificationCode AS
SELECT QualificationCode, QualificationName
FROM QUALIFICATION
GO
```

Name: ShowQualificationCodeOnApplicantId

```
CREATE PROCEDURE ksergis.ShowQualificationCodeOnApplicantId (@ApplicantId char(10))
AS
SELECT QUALIFICATION.QualificationCode, QualificationName
FROM QUALIFICATION_APPLICANT, QUALIFICATION
WHERE QUALIFICATION.QualificationCode = QUALIFICATION_APPLICANT.QualificationCode AND ApplicantId = @ApplicantId
GO
```

Name: ShowQualificationCodeOnJobId

```
CREATE PROCEDURE ksergis.ShowQualificationCodeOnJobId (@JobId char(10))
AS
SELECT distinct(JOB_QUALIFICATION.QualificationCode), QualificationName
FROM QUALIFICATION, JOB_QUALIFICATION
WHERE QUALIFICATION.QualificationCode = JOB_QUALIFICATION.QualificationCode AND JobId = @JobId
GO
```

Name: ShowQualificationNameOnJobId

```
CREATE PROCEDURE ksergis.ShowQualificationNameOnJobId (@JobId char(10))
AS
SELECT QualificationName
FROM JOB_QUALIFICATION, QUALIFICATION
WHERE JobId = @JobId AND JOB_QUALIFICATION.QualificationCode = QUALIFICATION.QualificationCode
GO
```

Name: ShowRankCode

```
CREATE PROCEDURE ksergis.ShowRankCode AS
SELECT RankCode, RankName
FROM RANK
GO
```

Name: ShowRankCodeOnJobId

```
CREATE PROCEDURE ksergis.ShowRankCodeOnJobId (@JobId char(10))
AS
SELECT distinct(JOB_RANK.RankCode), RankName
FROM RANK, JOB_RANK
WHERE RANK.RankCode =  JOB_RANK.RankCode AND JobId = @JobId
GO
```

Name: ShowRankCodeSpecialtyCodeSeaServiceOnApplicantId

```
CREATE PROCEDURE ksergis.ShowRankCodeSpecialtyCodeSeaServiceOnApplicantId (@ApplicantId char(10)) AS

SELECT RankCode, SpecialtyCode, SeaTimeForRank
FROM APPLICANT
WHERE APPLICANT.ApplicantId = @ApplicantId
GO
```

Name: ShowRankData

```
CREATE PROCEDURE ksergis.ShowRankData AS
SELECT *
FROM RANK
GO
```

Name: ShowRankNameTimeSeaServiceOnJobId

```
CREATE PROCEDURE ksergis.ShowRankNameTimeSeaServiceOnJobId (@JobId char(10))
AS
SELECT RankName, TimeSeaService
FROM JOB_RANK, RANK
WHERE JobId = @JobId AND JOB_RANK.RankCode = RANK.RankCode
GO
```

Name: ShowRankOnApplicantId

```
CREATE PROCEDURE ksergis.ShowRankOnApplicantId (@ApplicantId char(10)) AS
SELECT APPLICANT.RankCode, RankName
FROM APPLICANT, RANK
WHERE ApplicantId = @ApplicantId AND APPLICANT.RankCode = RANK.RankCode
GO
```

Name: ShowRankSpecialtySeaServiceOnApplicantId

```
CREATE PROCEDURE ksergis.ShowRankSpecialtySeaServiceOnApplicantId (@ApplicantId char(10)) AS

SELECT RankName, SpecialtyName, SeaTimeForRank
FROM APPLICANT, RANK, SPECIALTY
WHERE APPLICANT.ApplicantId = @ApplicantId AND APPLICANT.RankCode = RANK.RankCode AND APPLICANT.SpecialtyCode = SPECIALTY.SpecialtyCode
```

```
GO
```

Name: ShowSeaTimeForRankOnApplicantId

```
CREATE PROCEDURE ksergis.ShowSeaTimeForRankOnApplicantId (@ApplicantId char(10)) AS

SELECT ApplicantId, SeaTimeForRank
FROM APPLICANT
WHERE ApplicantId = @ApplicantId
GO
```

Name: ShowSpecialtyCode

```
CREATE PROCEDURE ksergis.ShowSpecialtyCode AS
SELECT SpecialtyCode, SpecialtyName
FROM SPECIALTY
GO
```

Name: ShowSpecialtyCodeOnJobId

```
CREATE PROCEDURE ksergis.ShowSpecialtyCodeOnJobId (@JobId char(10))
AS
SELECT distinct(JOB_SPECIALTY.SpecialtyCode), SpecialtyName
FROM SPECIALTY, JOB_SPECIALTY
WHERE SPECIALTY.SpecialtyCode =  JOB_SPECIALTY.SpecialtyCode AND JobId = @JobId
GO
```

Name: ShowSpecialtyNameOnJobId

```
CREATE PROCEDURE ksergis.ShowSpecialtyNameOnJobId (@JobId char(10))
AS
SELECT SpecialtyName
FROM JOB_SPECIALTY, SPECIALTY
WHERE JobId = @JobId AND JOB_SPECIALTY.SpecialtyCode = SPECIALTY.SpecialtyCode
GO
```

Name: ShowSpecialtyOnApplicantId

```
CREATE PROCEDURE ksergis.ShowSpecialtyOnApplicantId (@ApplicantId char(10))
AS

SELECT SPECIALTY.SpecialtyCode, SpecialtyName
FROM APPLICANT, SPECIALTY
```

```
WHERE ApplicantId = @ApplicantId AND APPLICANT.SpecialtyCode = SPECIALTY.SpecialtyCode
GO
```

Name: UpdateAddressData

```
CREATE PROCEDURE ksergis.UpdateAddressData (@ApplicantId char(10), @CityOrTown char(50), @Street char(50), @Appartment char(10), @ZIP char(10))
AS
UPDATE dbo.ADDRESS
SET CityOrTown = @CityOrTown, Street = @Street, Appartment = @Appartment, ZIP = @ZIP
WHERE ApplicantId = @ApplicantId
GO
```

Name: UpdateApplicantData

```
CREATE PROCEDURE ksergis.UpdateApplicantData (@ApplicantId char(10), @FirstName char(30), @LastName char(30), @MiddleName char(30), @EmailAddress char(50))
AS
UPDATE dbo.APPLICANT
SET FirstName = @FirstName, LastName = @LastName, MiddleName = @MiddleName, EmailAddress = @EmailAddress
WHERE ApplicantId = @ApplicantId
GO
```

Name: UpdateApplicantId

```
CREATE PROCEDURE ksergis.UpdateApplicantId (@ApplicantId char(10), @UserName char(50))
AS
UPDATE dbo.APPLICANT
SET ApplicantId = @ApplicantId
WHERE UserName = @UserName
GO
```

Name: UpdateApplicantIdSpecialtyRank

```
CREATE PROCEDURE ksergis.UpdateApplicantIdSpecialtyRank (@ApplicantId char(10),  @RankCode char(10), @SpecialtyCode char(10), @SeaTimeForRank float)
AS
UPDATE dbo.APPLICANT
SET RankCode = @RankCode, SpecialtyCode = @SpecialtyCode, SeaTimeForRank = @SeaTimeForRank
WHERE ApplicantId = @ApplicantId
```

GO

Name: UpdateCoefficient
CREATE PROCEDURE ksergis.UpdateCoefficient (@CoefficientId char(30), @CoefficientValue int) AS UPDATE COEFFICIENT SET CoefficientValue = @CoefficientValue WHERE CoefficientId = @CoefficientId GO

Name: UpdateCredentialsGrade
CREATE PROCEDURE ksergis.UpdateCredentialsGrade (@ApplicantId char(10), @CredentialsId char(10), @CredentialsGrade float) AS UPDATE dbo.APPLICANT_CREDENTIALS SET CredentialsGrade = @CredentialsGrade WHERE ApplicantId = @ApplicantId AND CredentialsId = @CredentialsId GO

Name: UpdateExperience
CREATE PROCEDURE ksergis.UpdateExperience (@JobId char(10), @ApplicantId char(10), @Experience float) AS UPDATE EXPERIENCE SET Experience = @Experience WHERE JobId = @JobId AND ApplicantId = @ApplicantId GO

Name: UpdateJobIdJobNameExperienceRequired
CREATE PROCEDURE ksergis.UpdateJobIdJobNameExperienceRequired (@JobId char(10), @JobIdNew char(10), @JobName char(30), @ExperienceRequired float) AS UPDATE dbo.JOB SET JobId = @JobIdNew, JobName = @JobName, ExperienceRequired = @ExperienceRequired WHERE JobId = @JobId GO

Name: UpdateJobNameExperienceRequired
CREATE PROCEDURE ksergis.UpdateJobNameExperienceRequired (@JobId char(10), @JobName char(30), @ExperienceRequired float)

```
AS
UPDATE dbo.JOB
SET JobName = @JobName, ExperienceRequired = @ExperienceRequired
WHERE JobId = @JobId
GO
```

Name: UpdateLanguageDegree
CREATE PROCEDURE ksergis.UpdateLanguageDegree (@ApplicantId char(10), @LanguageCode char(10), @LanguageDegree float) AS UPDATE dbo.APPLICANT_LANGUAGE SET LanguageDegree = @LanguageDegree WHERE ApplicantId = @ApplicantId AND LanguageCode = @LanguageCode GO

Name: UpdatePhoneData
CREATE PROCEDURE ksergis.UpdatePhoneData (@ApplicantId char(10), @HomePhoneNumber char(30), @CellPhoneNumber char(30), @OtherPhoneNumber char(30)) AS UPDATE PHONE SET HomePhoneNumber = @HomePhoneNumber, CellPhoneNumber = @CellPhoneNumber, OtherPhoneNumber = @OtherPhoneNumber WHERE ApplicantId = @ApplicantId GO

Name: UpdateUserNamePassword
CREATE PROCEDURE ksergis.UpdateUserNamePassword (@ApplicantId char(10), @UserName char(50), @Password char(50)) AS UPDATE dbo.APPLICANT SET UserName = @UserName, Password = @Password WHERE ApplicantId = @ApplicantId GO

Name: UpdateUserNamePasswordCommand
CREATE PROCEDURE ksergis.UpdateUserNamePasswordCommand (@CommandCode char(10), @UserName char(50), @Password char(50)) AS UPDATE dbo.COMMAND SET UserName = @UserName, Password = @Password WHERE CommandCode = @CommandCode GO

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. William R. Gates and Mark E. Nissen: Two Sided Matching Agents for Electronic Employment Market Design: Social Welfare Implications, December 2002

2. Hemant K. Bhargava and Kevin J. Snoap: Reengineering Recruit Distribution in the U.S. Marine Corps, October 28, 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Daniel R. Dolk
   Department of Information Sciences
   Naval Postgraduate School
   Monterey, California

4. Dr. Rudy Darken
   Department of Computer Science
   Monterey, California